

资深Android开发工程师倾心力作

循序渐进地介绍Android应用开发的核心技术，包括环境搭建、语言基础、布局及控件、四大组件、多媒体应用、数据处理技术、触摸和手势识别、多线程、网络技术、定位、蓝牙、VR和NDK开发等知识。
提供App完整项目案例，通过阅读本书，读者能够掌握Android应用开发所需要的各种技术，从0到1开发一款自己的App产品。





适用于Android 6/7与Android Studio 2.x

Android 开发实战 从学习到产品

李瑞奇 编著

清华大学出版社

Android开发实战 从学习到产品

李瑞奇 编著

清華大學出版社
北 京

内 容 简 介

本书由一线资深软件开发工程师基于目前广泛使用的 Android 6/7 和 Android Studio 2.x 开发环境倾力编撰，循序渐进地介绍了 Android 应用开发的主要内容，包括开发环境搭建、Android 语言基础、常用布局及控件、四大组件、图形图像技术、多媒体应用、数据处理技术、触摸和手势识别、多线程、网络技术、定位、蓝牙以及 VR 和 NDK 开发等知识，全书代码示例丰富，提供 App 完整项目案例，通过阅读本书，读者能够掌握 Android 应用开发所需要的各种技术和从 0 到 1 开发一款自己的 App 产品。

本书适合于 Android 初学者、移动开发从业者学习，也可作为培训机构及职业学院软件开发实践课的参考教材。

图书在版编目（CIP）数据

Android 开发实战：从学习到产品/李瑞奇编著. —北京：清华大学出版社，2017
ISBN 978-7-302-46802-8

Ⅰ. ①A… Ⅱ. ①李… Ⅲ. ①移动终端—应用程序—程序设计 Ⅳ. ①TN929.53

中国版本图书馆 CIP 数据核字（2017）第 052717 号

责任编辑：王金柱
封面设计：王　翔
责任校对：闫秀华
责任印制：

出版发行：清华大学出版社
网　　址：http://www.tup.com.cn，http://www.wqbook.com
地　　址：北京清华大学学研大厦 A 座　　**邮　　编**：100084
社 总 机：010-62770175　　**邮　　购**：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质 量 反 馈：010-62772015，zhiliang@tup.tsinghua.edu.cn

印 装 者：
经　　销：全国新华书店
开　　本：190mm×260mm　　**印　　张**：30.75　　**字　　数**：788 千字
版　　次：2017 年 5 月第 1 版　　**印　　次**：2017 年 5 月第 1 次印刷
印　　数：1~3000
定　　价：79.00 元

产品编号：070543-01

前　　言

编写本书的目的

随着 Android 系统的迅猛发展，它已经成为全球范围内具有广泛影响力的操作系统，越来越多的厂商加入到 Android 的阵营，至 2017 年 1 月，Google 公司对外公布，其旗下所属的 Android 系统全球市场占有率已经高达 90%。各大中小型手机制造商近些年都在引入 Android 工程师，开发基于 Android 系统的智能手机。Android 系统早就不仅仅是一款手机的操作系统，越来越广泛地应用于平板电脑、可佩戴设备、电视、数码相机等，造就了目前 Android 开发人才需求的快速增长。从大趋势上看，Android 软件人才的需求将越来越大。

在这种背景下，Android 开发学习者的队伍渐渐庞大起来，但是市场上适合 Android 开发者学习使用的书籍虽然并不少，但大多版本都已过时，有很多还是基于 Android 4/5 编写的，甚至有一些是基于 Android 2.3 的。Android 发展到今天，已经推出了 7.0 版本，使用旧版本书籍进行学习会有诸多问题，严重时甚至会使读者开发的应用崩溃。另一个比较重要的问题是，几乎所有书籍使用的 IDE 都是 Eclipse 加 ADT 插件，但是，在大部分企业中 Android 开发早已使用 Android Studio 作为 IDE 了。这些都导致一些书籍的实用性大大下降。

本书由一线资深软件开发工程师基于目前广泛使用的 Android 6/7 和 Android Studio 2.x 开发环境倾力编撰，旨在帮助 Android 初学者和开发人员尽快掌握在 Android Studio 环境下进行应用开发的方法和技术。

本书主要内容

本书共 15 章，各章内容说明如下：

第 1 章对 Android 的发展史与现状和 Android 系统的特性做简单介绍，讲解如何搭建 Android 开发环境，并介绍如何使用 Android Studio 来创建第一个 Android 程序。

第 2 章通过一个工程实例来阐述 Android App 是如何运行的，并引出 Activity 这一在 Android 开发中极其重要的组件。在本章中，系统地讲解了 Activity 的概念、生命周

期、多个 Activity 之间的跳转，以及 Activity 的 4 种启动模式。另外，本章还介绍 Intent 在 Activity 组件中的应用，并且讲述如何使用 Log。

第 3 章主要介绍布局管理器的作用，并介绍 Android 中的 6 种布局管理器，即 LinearLayout、RelativeLayout、TableLayout、FrameLayout、AbsoluteLayout、GridLayout。所有的布局管理器既可以通过配置文件实现，也可以在 Activity 中用代码实现。布局管理器直接可以通过互相嵌套使用来实现更复杂的布局。

第 4 章系统地讲解在 Android 开发中常用的一些控件，同时结合控件讲解 Android 中的事件处理，对实际开发中经常使用的控件 ListView 进行了重点讲解。

第 5 章系统地讲述 Fragment 的使用场景、使用方法和生命周期，并将其与 Activity 的生命周期做比较，以便加深对 Fragment 的理解。同时，对 ListFragment 与 DialogFragment 这两个特殊的 Fragment 进行深入的讲解，对其用法和特性也都进行了分析。在本章最后还根据开发中的经验向读者阐释一些 Fragment 使用中常见的问题。

第 6 章非常详细地讲述 ViewPager、RecyclerView 这两个 View 控件的使用。这两个控件都是比较新的控件，在已有的 Android 开发书籍中很少提及，而在实际的开发过程中又经常使用，所以这里花较多篇幅对其讲解。同时，针对一些特殊情况，比如官方提供的控件无法解决的问题，如何通过自定义控件来解决也进行了讲解。

第 7 章主要讲解数据操作的内容，系统地讲述 4 种数据存储的具体方式。同时，本章引入动态权限的概念，提醒读者在使用 Android 6.0 以上版本进行开发时，添加权限应该是动态获取，而不是静态获取。

第 8 章讲解 Service 是什么、Service 的分类、为什么需要使用 Service 以及 Service 的几种使用方法，同时结合 Service 讲解 Handler 机制和 AsyncTask 的用法。

第 9 章阐述广播机制，并通过实例告诉读者如何使用系统广播，以及通过对普通广播和有序广播的介绍讲解如何自定义广播。另外，本章还讲述 Android 为了能够简单地解决广播的安全性问题而引入的一套本地广播机制——本地广播。

第 10 章对 Android 中的网络通信技术进行系统的分析与总结，讲解如何使用 HTTP 及 Socket 进行网络通信，同时针对一些特殊的需要讲解 WebView 的使用，重点介绍 OkHttp 这一实际开发中经常使用的、非常重要的 HTTP 请求框架。

第 11 章主要对 Android 系统中的各种多媒体技术进行学习，其中包括通知的使用技巧、调用摄像头拍照、从相册中选取照片、播放音频和视频文件，以及如何进行视频和音频的录制。此外，本章还介绍如何使用 Android 提供的 API 来接收、发送和拦截短信，这使得读者甚至可以编写一个自己的短信程序来替换系统的短信程序。

第 12 章主要以传感器和地理信息技术为例讲解 Android 中具有特色的一些功能：传感器和地理信息技术。具体来说就是介绍加速度传感器、光照传感器、方向传感器的使用，并根据它们的原理开发具有特殊功能的小应用；以及通过使用地理信息技术开发能够定位的应用，使用 Geocoder 类进行地理位置解析、获取具体的位置，通过使用第三方工具高德地图来展示位置。

第 13 章主要介绍 VR 这一热门技术，阐述 VR 的技术实现原理、存在的瓶颈以及当前的市场现状和市场前景，最后通过一个实例来讲解基于 unity3D 的 Android 平台 VR 应用开发。

第 14 章讲述 Android NDK 开发的背景以及优势，并详细讲解如何使用 Android Studio 进行 Android NDK 开发。

第 15 章通过一个完整的应用讲述在开发实践中如何将一个产品从需求变为实际可用的应用，并将其发布到应用市场。

本书适合的读者

本书详细地介绍 Android 开发的各种知识和技术，从基础到实践，提供了大量代码示例和完整的项目案例，无论是初次接触 Android 开发的读者，还是想提高 Android 开发技能的程序员，包括大学生和企业互联网营销人员，都可以通过本书获益。

由于笔者水平有限，书中难免有欠妥之处，敬请广大读者批评指正。对于书中存在的问题，读者若有什么建议或意见可发信至 527409323@qq.com，编者会在第一时间回复。

本书示例源代码下载

为了方便读者学习，本书提供了对应的范例程序，下载地址为

http://pan.baidu.com/s/1skOP8PB（区分英文字母大小写以及数字和字母）

如果下载有问题，请电子邮件联系 booksaga@126.com，邮件主题为“Android 开发实战：从学习到产品”。

致谢

编者的很多知识都来源于互联网。互联网是一个丰富的知识资源库，只要你愿意探索总能获得有用的东西。感谢那些在互联网上免费分享知识资源的人们，是他们丰富了互联网的内涵，发扬了知识共享的精神，使得每个人可以平等地获取知识、得到进步。

感谢 Google 公司和它的 Android 开发团队，可以说是他们创造了这个移动互联网时代。

感谢清华大学出版社王金柱编辑的支持和鼓励，感谢他在本书编写与出版过程中的热情帮助和耐心指导。

编　者
2017 年 2 月 8 日

目　　录



第 1 章 初识 Android

对于 Android 的初学者来说，对 Android 开发还是很陌生的，因此本章的重点就是向读者介绍 Android 的过去与现在，并对 Android 的系统架构做详细的介绍。同时，本章还将讲解如何搭建使用 Android Studio 作为 IDE（集成开发环境）的 Android 开发环境，这是开发的基础，是应该熟练掌握的。本章最后通过一个简单的 Android 项目来展示 Android Studio 的基本使用、常见问题以及 Android 工程的基本目录。

1.1 Android 发展史与现状

2003 年 10 月，Andy Rubin 等人创建了与 Android 系统同名的 Android 公司，并组建了 Android 开发团队，最初的 Android 系统是一款针对数码相机开发的智能操作系统，之后被 Google 公司低调收购，并聘任 Andy Rubin 为 Google 公司工程部副总裁，继续负责 Android 项目。

自 Android 系统首次发布至今，Android 经历了很多的版本更新。表 1-1 列出了 Android 系统不同版本的发布时间及对应的版本号。

表 1-1　Android 各版本发布时间及代号

Android 版本	发布日期	代号
Android 1.1		
Android 1.5	2009 年 4 月 30 日	Cupcake（纸杯蛋糕）
Android 1.6	2009 年 9 月 15 日	Donut（炸面圈）
Android 2.0/2.1	2009 年 10 月 26 日	Eclair（长松饼）
Android 2.2	2010 年 5 月 20 日	Froyo（冻酸奶）
Android 2.3	2010 年 12 月 6 日	Gingerbread（姜饼）
Android 3.0/3.1/3.2	2011 年 2 月 22 日	Honeycomb（蜂巢）
Android 4.0	2011 年 10 月 19 日	Ice Cream Sandwich（冰淇淋三明治）
Android 4.1	2012 年 6 月 28 日	Jelly Bean（果冻豆）
Android 4.2	2012 年 10 月 8 日	Jelly Bean（果冻豆）
Android 5.0	2014 年 10 月 15 日	Lime Pie（酸橙派）
Android 6.0	2015 年 5 月 28 日	Marshmallow（棉花糖）
Android 7.0	2016 年 3 月 10 日	Nougat(牛轧糖)

从 Android 1.5 版本开始，Android 系统越来越像一个智能操作系统，Google 开始将 Android 系统的版本以甜品的名字命名。随着 Android 系统近年来的快速普及与发展，越来越多的厂商加入到 Android 的阵营，至 2016 年 5 月，Google 公司对外公布，其旗下所属的 Android 系统全球市场占有率已经高达 85%。

Android 系统是基于 Linux 的智能操作系统，2007 年 11 月，Google 与 84 家硬件制造商、软件开发商及电信运营商组建开发手机联盟，共同研发改良 Android 系统。随后 Google 以 Apache 开源许可证的授权方式发布了 Android 的源代码。也就是说 Android 系统是完整公开并且免费的，它的快速发展与这一点有很大关系。

1.2　Android 系统架构与特性

Android 是什么？就像 Android 开源和兼容性技术负责人 Dan Morrill 在 Android 开发手册兼容性部分所解释的，“Android 并不是传统的 Linux 风格的一个规范或分发版本，也不是一系列可重用的组件集成，Android 是一个用于连接设备的软件块。”Android 是一个软件系统，用于连接设备，并不是大家平时所说的操作系统。

1.2.1　Android 系统架构

Android 的系统架构和其他操作系统一样，采用了分层的架构。从图 1-1 所示的架构图看，Android 分为 4 层，从高层到低层分别是应用程序层（Application）、应用程序框架层（Application Framework）、系统运行库层（Libraries）和 Linux 内核层（Linux Kernel）。Android 操作系统可以在 4 个主要层面上分为 5 部分。

图 1-1　系统架构图

1. 应用程序层

Android 系统包含了一系列核心应用程序，包括电子邮件、短信 SMS、日历、拨号器、地图、浏览器、联系人等。这些应用程序都是用 Java 语言编写的。本书重点讲解如何编写 Android 系统上运行的应用程序，在程序分层上，与系统核心应用程序平级。

2. 应用程序框架层

Android 应用程序框架提供了大量的 API，以供开发人员使用。Android 应用程序的开发就是调用这些 API，根据需求实现功能。

应用程序框架是应用程序的基础。为了软件的复用，任何一个应用程序都可以开发 Android 系统的功能模块，只要发布的时候遵循应用程序框架的规范，其他应用程序也可以使用这个功能模块。

3. 系统运行库层

Android 系统运行库是用 C/C++语言编写的，是一套被不同组件所使用的函数库组成的集合。一般来说，Android 应用开发者无法直接调用这套函数库，都是通过上层的应用程序框架提供的 API 来对这些函数库进行调用。

下面对一些核心库进行简单的介绍。

- Libc: 从 BSD 系统派生出来的标准 C 系统库，在标准 C 系统库基础之上为便携式 Linux 系统专门进行了调整。
- Medio Framework: 基于 PacketView 的 OpenCORE，这套媒体库支持播放与录制硬盘及视频格式的文件，并能查看静态图片。
- Surface Manager: 在执行多个应用程序时，负责管理显示与存取操作间的互动，同时负责 2D 绘图与 3D 绘图进行显示合成。
- WebKit: Web 浏览器引擎，为 Android 浏览器提供支持。
- SGL: 底层的 2D 图像引擎。
- 3D libraries: 基于 OpenGL ES 1.0 API，提供使用软硬件实现 3D 加速的功能。
- FreeType: 提供位图和向量字体的支持。
- SQLite: 轻量级的关系型数据库。

4. Android 运行时

Android 运行时由两部分完成：Android 核心库和 Dalvik 虚拟机。其中核心库集提供了 Java 语言核心库所能使用的绝大部分功能，Dalvik 虚拟机负责运行 Android 应用程序。

虽然 Android 应用程序通过 Java 语言编写，并且每个 Java 程序都会在 Java 虚拟机 JVM 内运行，但是 Android 系统毕竟是运行在移动设备上的，由于硬件的限制， Android 应用程序并不使用 Java 的虚拟机 JVM 来运行，而是使用自己独立的虚拟机 Dalvik VM（针对多个同时高效运行的虚拟机进行了优化）。每个 Android 应用程序都运行在单独的一个 Dalvik 虚拟机内，因此 Android 系统可以方便地对应用程序进行隔离。

5. Linux 内核

Android 系统是基于 Linux 2.6 之上建立的操作系统。Linux 内核为 Android 系统提供了安全性、内存管理、进程管理、网络协议栈、驱动模型等核心系统服务。Linux 内核帮助 Android 系统实现了底层硬件与上层软件之间的抽象。

1.2.2 Dalvik VM 和 JVM 的区别

JVM（Java 虚拟机）是一个虚构出来的运行 Java 程序的运行时，是通过在实际的计算机上仿真模拟各种计算机功能的实现。它具有完善的硬件架构（如处理器、堆栈、寄存器等），还具有相应的指令系统，使用 JVM 就是使 Java 程序支持与操作系统无关。理论上在任何操作系统中，只要有对应的 JVM，即可运行 Java 程序。

Dalvik VM 是在 Android 系统上运行 Android 程序的虚拟机，其指令集是基于寄存器架构的，执行特有的文件格式-dex 字节码来完成对象生命周期管理、堆栈管理、线程管理、安全异常管理、垃圾回收等重要功能。

由于 Android 应用程序的开发编程语言是 Java，而 Java 程序运行在 JVM（Java 虚拟机）上，因此有些人会混淆 Android 的虚拟机 Dalvik VM 和 JVM，但是实际上 Dalvik 并未遵守 JVM 规范，而且两者也是互不兼容。

Dalvik VM 和 JVM 的编译过程如下：

- JVM：.java→.class→.jar
- Dalvik VM：.java→.class→.dex

从它们的编译过程可以看出，JVM 运行的是.class 文件的 Java 字节码，但是 Dalvik VM 运行的是其转换后的 dex（Dalvik Executable）文件。JVM 字节从.class 文件或者 JAR 包中加载字节码然后运行，而 Dalvik VM 无法直接从.class 文件或 JAR 包中加载字节码，需要通过 DX 工具将应用程序所有的.class 文件编译成一个.dex 文件后再运行。

如图 1-2 显示了 Dalvik VM 与 JVM 编译过程的区别。

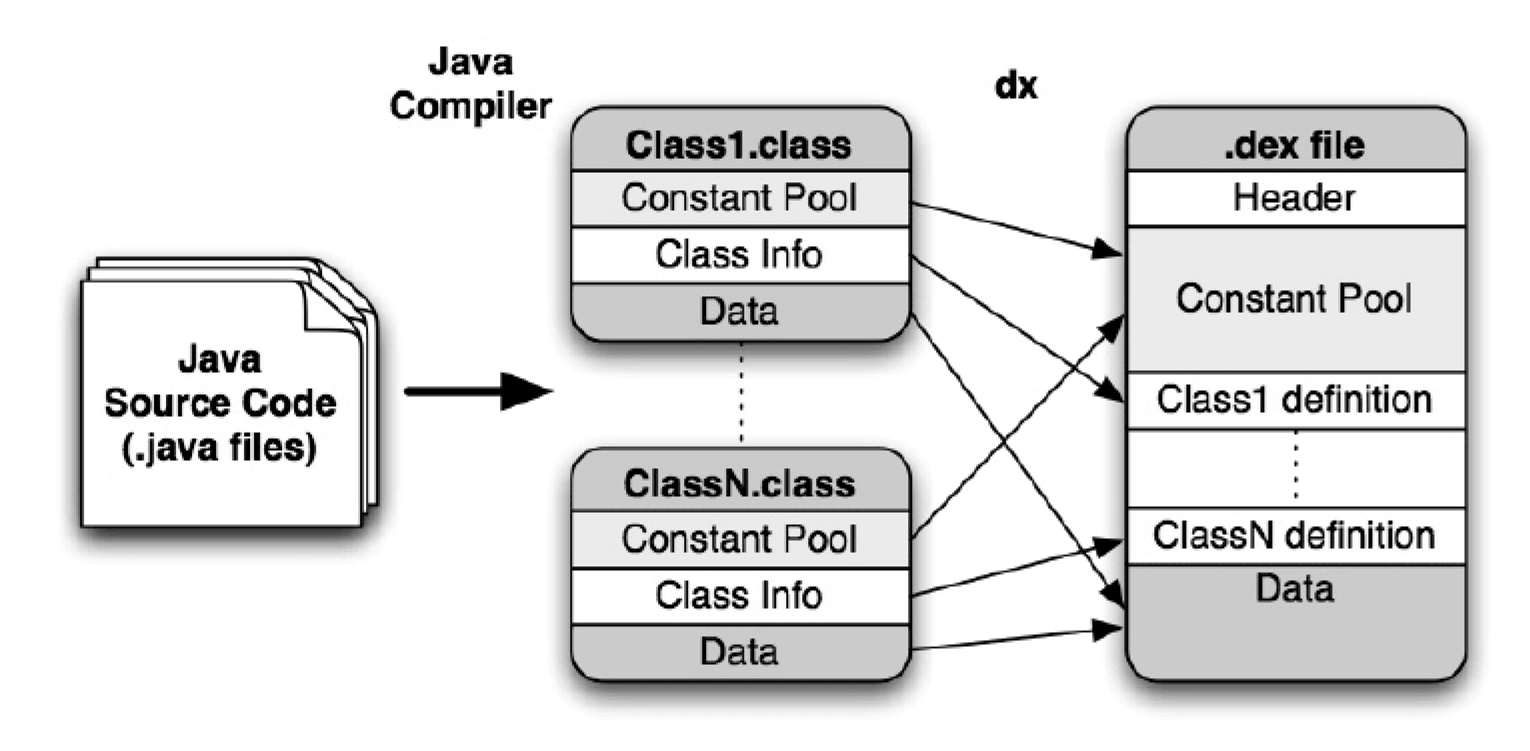


图 1-2 Dalvik VM 与 JVM 编译过程的区别

从图 1-2 中可以看出，Dalvik VM 把.java 文件编译成.class 后会对.class 进行重构，整合基本元素（常量池、类定义、数据段），最后压缩写进一个.dex 文件中。其中，常量池描述了所有的常量，包括引用、方法名、数值常量等；类定义包括访问标识、类名等基本信息；数据段中包含各种被 VM 指定的方法代码以及类和方法的相关信息和实例变量。这种把多个.class 文件进行整合的方法大大提高了 Android 程序的运行速度，例如应用程序中多个类定义了字符串常量 TAG，而在 JVM 中会编译成多个.class 文件，每个.class 文件的常量池中均包含这个 TAG 常量，但是 Dalvik VM 在编译成.dex 文件之后，其常量池里只有一个 TAG 常量。

JVM 和 Dalvik VM 还有一点非常重要的差异，就是基于的架构不同。JVM 是基于栈的架构，而 Dalvik VM 是基于寄存器的架构。相对于基于栈的 JVM 而言，基于寄存器的 Dalvik VM 实现虽然牺牲了一些硬件上的通用性，但是在代码的执行效率上要更胜一筹。一般来讲，VM 中指令的解释执行时间主要花费在以下 3 个方面：

- 分发指令。
- 访问运算数。
- 执行运算。

其中，分发指令这个环节对性能的影响最大。在基于寄存器的 Dalvik VM 中可以更有效地减少冗余指令的分发，减少内存的读写访问。

从 JVM 和 Dalvik VM 的区别上来说，Dalvik VM 主要是针对 Android 这个嵌入式操作系统的特点进行各种优化，使其更省电、更省内存、运行效率更高，但是牺牲了一些 JVM 与平台无关的特性。实际上，Dalvik VM 本身就是为 Android 设计的，无须考虑其他平台的问题。这里只介绍 JVM 和 Dalvik VM 的两个重要区别，因为本书并不是讲解 Android 内核的，所以只点明了 Dalvik VM 的特点。读者对这部分的内容了解即可。

1.2.3 Android 系统平台的优势

Android 系统相对于其他操作系统，有如下几点优势。

1. 开放性

首先就是 Android 系统的开放性，其开发平台允许任何移动终端厂商加入 Android 联盟，降低了开发门槛，使其拥有更多的开发者，随着用户和应用的日益丰富，也将推进 Android 系统的成熟。同时，开放性有利于 Android 设备的普及以及市场竞争力，有利于消费者买到更低价位的 Android 设备。

2. 丰富的硬件选择

同样由于 Android 系统的开放性，众多硬件厂商可以推出各种搭载 Android 系统的设备。现如今，Android 系统不仅仅运行在手机上，越来越多的设备开始支持 Android 系统，如电视、可佩戴设备、数码相机等。

3. 便于开发

Google 开放了 Android 的系统源码，给开发者提供了一个自由的开发环境，不必受到各种条条框框的束缚。

4. Google 服务的支持

Google 公司作为一个做服务的公司，提供了地图、邮件、搜索等服务。Android 系统可以对这些服务进行无缝结合。

1.3 Android 开发环境搭建

现在的 Android 开发环境有两种，一种是基于 Eclipse+ADT（Android 开发者工具）的开发环境，另一种是基于 Android Studio 的开发环境。目前，基于 Eclipse+ADT 的开发环境已经很少使用，主流是基于 Android Studio 的开发环境。本书的所有开发都是使用 Android Studio 进行的。

Android Studio 是 Google 开发的一款面向 Android 开发者的 IDE，支持 Windows、Mac、Linux 等操作系统，基于流行的 Java 语言集成开发环境 IntelliJ 搭建而成。该 IDE 在 2013 年 5 月的 Google I/O 开发者大会上首次露面，当时的测试版有各种莫名其妙的 Bug。2014 年 12 月 8 日发布了稳定版，自 Android Studio 1.0 推出后，Google 官方逐步放弃了对 Eclipse ADT 的支持，并为 Eclipse 用户提供了工程迁移的解决办法。与 Eclipse+ADT 相比，Android Studio 有很多优势：

（1）Android Stuido 是 Google 推出、专门为 Android“量身订做”的，是 Google 大力支持的一款基于 IntelliJ idea 改造的 IDE，Google 的工程师团队肯定会不断完善，上升空间非常大，这个应该能说明为什么它是 Android 的未来。

（2）Eclipse 的启动速度、响应速度、内存占用一直被诟病，而且经常遇到卡死状态。Studio 在这几个方面都全面领先 Eclipse。

（3）更加智能，提示补全对于开发来说意义重大，有了智能保存就再也不用每次都按 Ctrl + S 键了。熟悉 Studio 以后效率会大大提升。

（4）整合了 Gradle 构建工具。Gradle 是一个新的构建工具，Studio 天然支持 Gradle。Gradle 集合了 Ant 和 Maven 的优点，不管是配置、编译还是打包都非常优秀。

（5）Android Studio 的编辑器非常智能，除了吸收 Eclipse+ADT 的优点之外，还自带了多设备的实时预览。

（6）Studio 内置终端，对习惯命令行操作的人来说是一个好消息，再也不用来回切换了，一个 Studio 即可全部搞定。

（7）安装的时候就自带了 GitHub、Git、SVN 等流行的版本控制系统，可以直接 check out 项目。

Android 开发是使用 Java 的，所有不管是用什么方式搭建 Android 开发环境，都需要先配置 Java 环境。因此搭建基于 Android Studio 的 Android 开发环境分为两步，第一步是搭建 Java 环境，第二步是安装 Android Studio 以及 Android SDK。

1.3.1 下载安装 Java 并配置环境变量

首先我们需要下载 Java 开发工具包 JDK，下载地址为 http://www.oracle.com/technetwork/java/javase/downloads/index.html。在下载页面中选择接受许可，并根据系统选择对应的版本（本文以 Window 64 位系统为例），如图 1-3 所示。

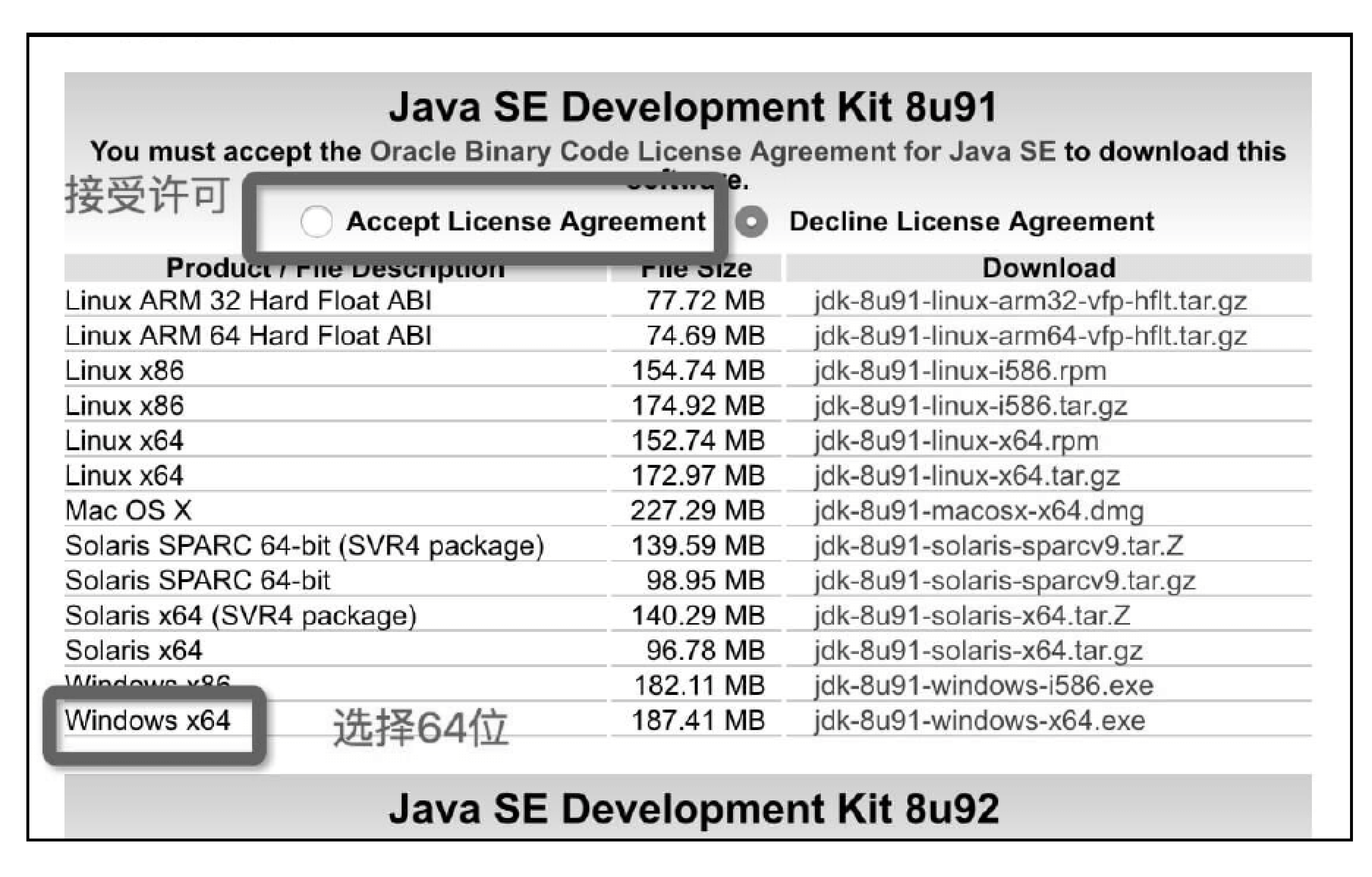


图 1-3　下载 Java 开发工具包 JDK

完成下载后 JDK 的安装根据提示进行，安装 JDK 的时候也会安装 JRE，一并安装就可以了。安装 JDK 过程中可以自定义安装目录等信息，例如我们选择安装目录为 C:\Program Files (x86)\Java\jdk1.8.0_91（这里面的路径，读者可以根据需要自行设置。需注意的是，不能含有中文字符）。

安装完成后，需要配置环境变量。

右击“我的电脑”，单击“属性”，选择“高级系统设置”，如图 1-4 所示。



图 1-4　选择“高级系统设置”

选择“高级”选项卡，单击“环境变量”按钮，如图 1-5 所示，然后就会出现如图 1-6 所示的界面。

在“系统变量”中设置 3 项属性，即 JAVA_HOME、PATH、CLASSPATH（不区分大小写），若已存在则单击“编辑”按钮，不存在则单击“新建”按钮。



图 1-5 单击“环境变量”按钮



图 1-6 “环境变量”对话框

变量参数设置（见图 1-7～图 1-9）如下：

- 变量名：JAVA_HOME
  变量值：C:\Program Files (x86)\Java\jdk1.8.0_91（要根据自己的实际路径配置）
- 变量名：CLASSPATH
  变量值：.;%JAVA_HOME%\lib\dt.jar;%JAVA_HOME%\lib\tools.jar;（注意前面有个“.”）
- 变量名：Path
  变量值：%JAVA_HOME%\bin;%JAVA_HOME%\jre\bin;



图 1-7 设置 JAVA_HOME



图 1-8 设置 PATH

配置完成后，可以通过命令行窗口测试是否配置成功。通过“开始”→“运行”命令打开“运行”对话框，输入“cmd”后打开命令行窗口。输入命令“java”，出现如图 1-10 所示的信息。

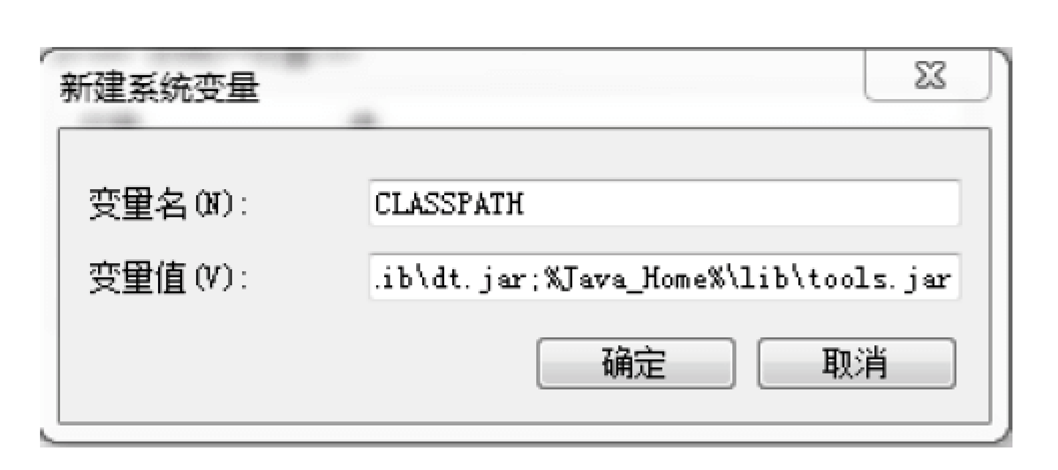


图 1-9　设置 CLASSPATH



```
C:\Users\Administrator>java
用法: java [-options] class [args...]
           (执行类)
   或  java [-options] -jar jarfile [args...]
           (执行 jar 文件)
其中选项包括:
    -d32          使用 32 位数据模型 (如果可用)
    -d64          使用 64 位数据模型 (如果可用)
    -client       选择 "client" VM
    -server       选择 "server" VM
                  默认 VM 是 client.

    -cp <目录和 zip/jar 文件的类搜索路径>
    -classpath <目录和 zip/jar 文件的类搜索路径>
                  用 ; 分隔的目录, JAR 档案
                  和 ZIP 档案列表, 用于搜索类文件。
```

图 1-10　在 DOS 界面输入“java”出现的信息

输入“javac”出现如图 1-11 所示的信息。

```
C:\Users\Administrator>javac
用法: javac <options> <source files>
其中, 可能的选项包括:
  -g                         生成所有调试信息
  -g:none                    不生成任何调试信息
  -g:{lines,vars,source}     只生成某些调试信息
  -nowarn                    不生成任何警告
  -verbose                   输出有关编译器正在执行的操作的消息
  -deprecation               输出使用已过时的 API 的源位置
  -classpath <路径>            指定查找用户类文件和注释处理程序的位置
  -cp <路径>                   指定查找用户类文件和注释处理程序的位置
  -sourcepath <路径>           指定查找输入源文件的位置
  -bootclasspath <路径>        覆盖引导类文件的位置
```

图 1-11　在 DOS 界面输入“javac”出现的信息

输入“java　-version”出现如图 1-12 所示的信息（和下载的版本号一致）。

```
C:\Users\Administrator>java -version
java version "1.8.0_101"
Java(TM) SE Runtime Environment (build 1.8.0_101-b13)
Java HotSpot(TM) Client VM (build 25.101-b13, mixed mode, sharing)

C:\Users\Administrator>
```

图 1-12　在 DOS 界面输入“java　- version”测试版本信息

如果上述信息都没有问题，就说明 Java 环境已经搭建完成了。

1.3.2　下载安装 Android Studio 和 Android SDK

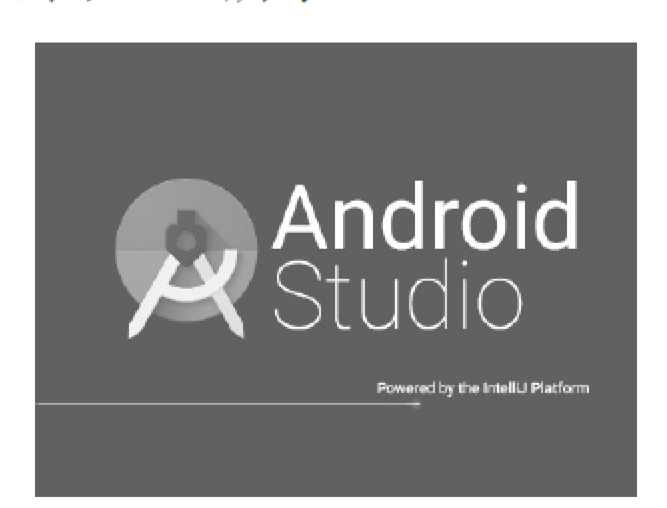
Android Studio 安装包分为含 Android SDK 版本和不含 Android SDK 两版本，如果已经下载了 SDK，那么完全可以下载不含 SDK 版本；如果下载了含 SDK 版本，那么既可以安装时选择自定义 SDK，也可以安装后重新指定 SDK 路径。这里我们下载安装含 SDK 版本的 Android Studio。

下载 Android Studio 需要访问 Google 官网，由于一些众所周知的原因，通过正常途径是访问不了的，虽然可以通过 VPN 来访问下载，不过这样的速度比较慢，因此建议读者通过国内的开源站下载。或者直接百度“Android Studio”，利用搜索页面上提供的 Android Studio

下载链接直接下载。这是百度提供的下载源，速度较快，而且下载包内直接包括了 Android SDK。下载过程这里不做演示。

下载完成后，双击文件安装。整个安装过程很简单，大部分只需要单击 Next 或者 Agree 按钮即可。下载的 Android Studio 是集成了 Android SDK 的，所以在安装过程中，遇到选择插件时记得勾选上 Android SDK。

安装好了以后，首次运行 Android Studio 一般都是可以成功的。Android Studio 的启动过程如图 1-13 所示。



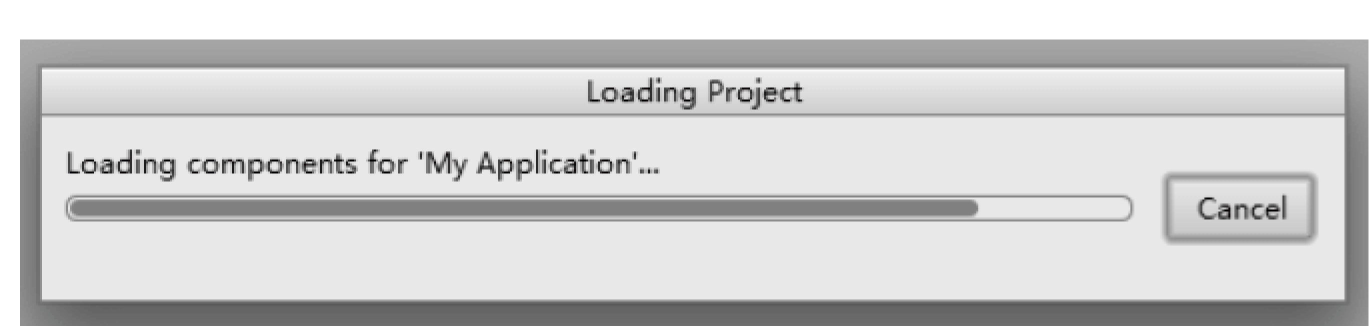


图 1-13 Android Studio 的启动过程

第一次启动 Android Studio 时需要设置 SDK 的安装目录，因此会弹出如图 1-14 所示的对话框，选择安装时的安装目录就可以了。

打开 Android Studio 之后会进入一个新建项目或者打开已有项目的选择界面，如图 1-15 所示。



图 1-14 选择安装目录

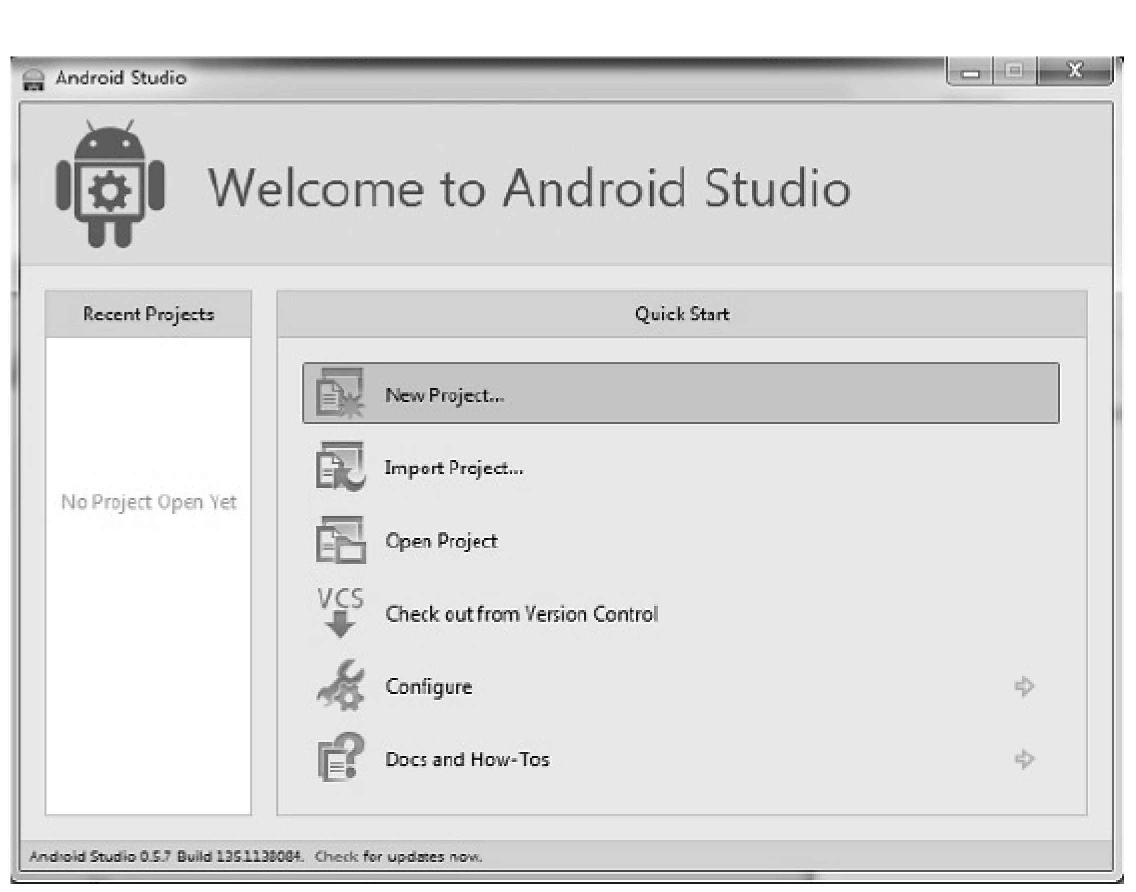


图 1-15 Android Studio 的欢迎界面

如果顺利地到达此步骤，就说明安装成功了。但是也有一种情况，启动界面会一直停在 Fetching Android SDK component information（见图 1-16）界面。

这是由于众所周知的一些原因导致的，比如谷歌公司在国内没有服务器、长城防火窗的存在（我国对因特网内容进行自动审查和过滤

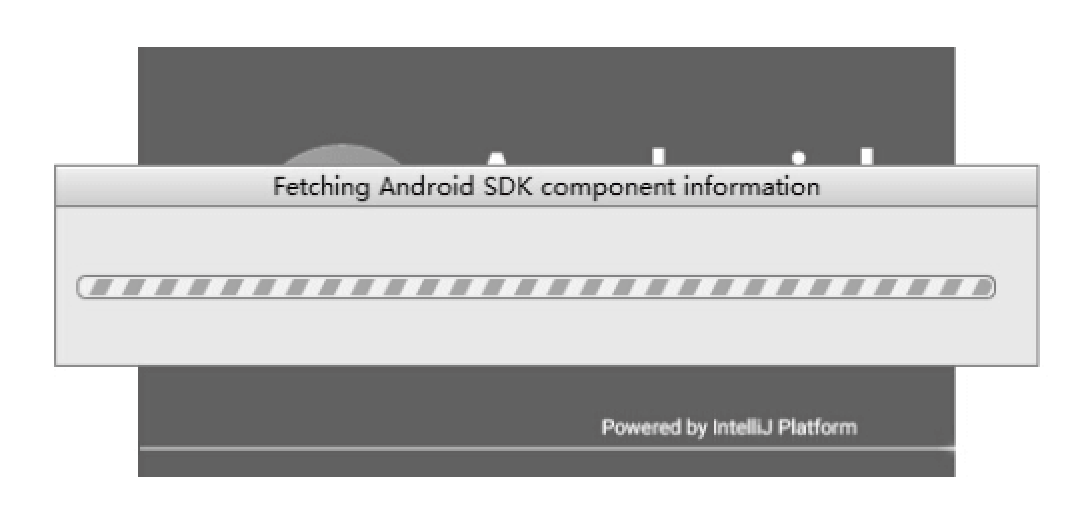


图 1-16 更新 SDK 被防火窗阻拦的停留界面

监控、由计算机与网络设备等软硬件所构成的系统）等。国内访问国外网络时会受到非常大的限制。解决办法就是关闭安装向导，如果无法关闭就在任务管理器中手动关掉进程（按 Ctrl+Alt+Del 组合键启动任务管理器），然后打开 Android Studio 安装目录下 bin 目录里的 idea.properties 文件，添加一条禁用开始运行向导的配置项：

```
disable.android.first.run=true
```

然后启动程序，就会打开项目向导界面。这时单击 Start a new Android Studio project 是没有反应的，并且在 Configure 下面的 SDK Manager 是灰色的——因为没有安装 Android SDK。这时一般可以采用以下两种做法：

- 没有 SDK 时，需要从网络下载。打开向导的 Configure-Settings，在查找框里面输入 proxy，找到下面的 HTTP Proxy，设置代理服务器，并且将 Force https://… sources to be fetched using http://选中，然后退出。将上面在 idea.properties 配置文件中添加的那条配置项注释掉。重新打开 Android Studio，等把 Android SDK 下载安装完成就可以了。
- 有 SDK，重新指定 SDK 路径。打开向导的 Configure→Project Defaults→Project Structure，在此填入已有的 SDK 路径。

重启 Android Studio 就可以在向导里新建 Android 工程了，至此整个安装过程结束。

1.4 Android Studio 的使用与工程目录解析

完成了 Android 开发环境的安装之后就可以进行 Android 工程的开发了。本节我们将创建第一个 Android 应用，并通过这个应用的创建来介绍 Android Studio 开发环境的使用。同时，还将向读者介绍 Android 工程目录的内容。

1.4.1 建立新的 Android 应用

新建工程，输入工程名、主包名和存储路径，如图 1-17 所示。

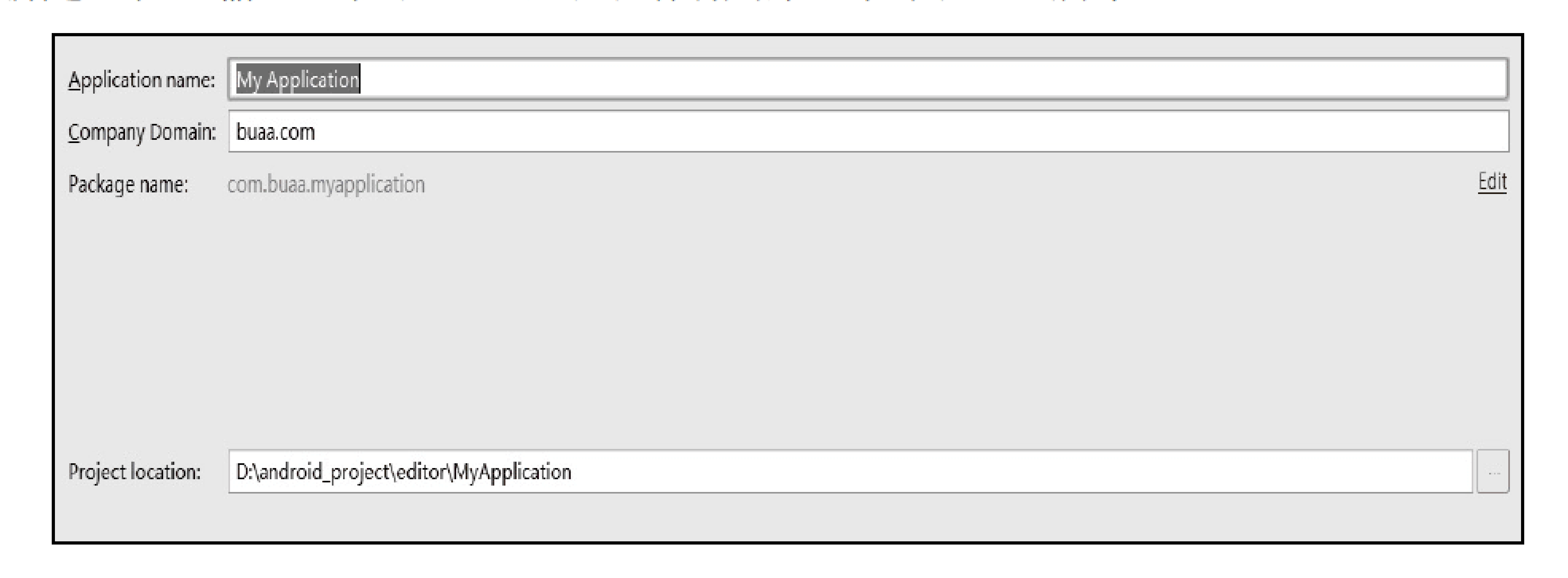


图 1-17　输入工程名、主包名和存储路径

连续单击 Next 按钮一直到如图 1-18 所示的步骤，在此处选择 App 要适配的设备（Wear、Phone and Tablet 或 TV）。

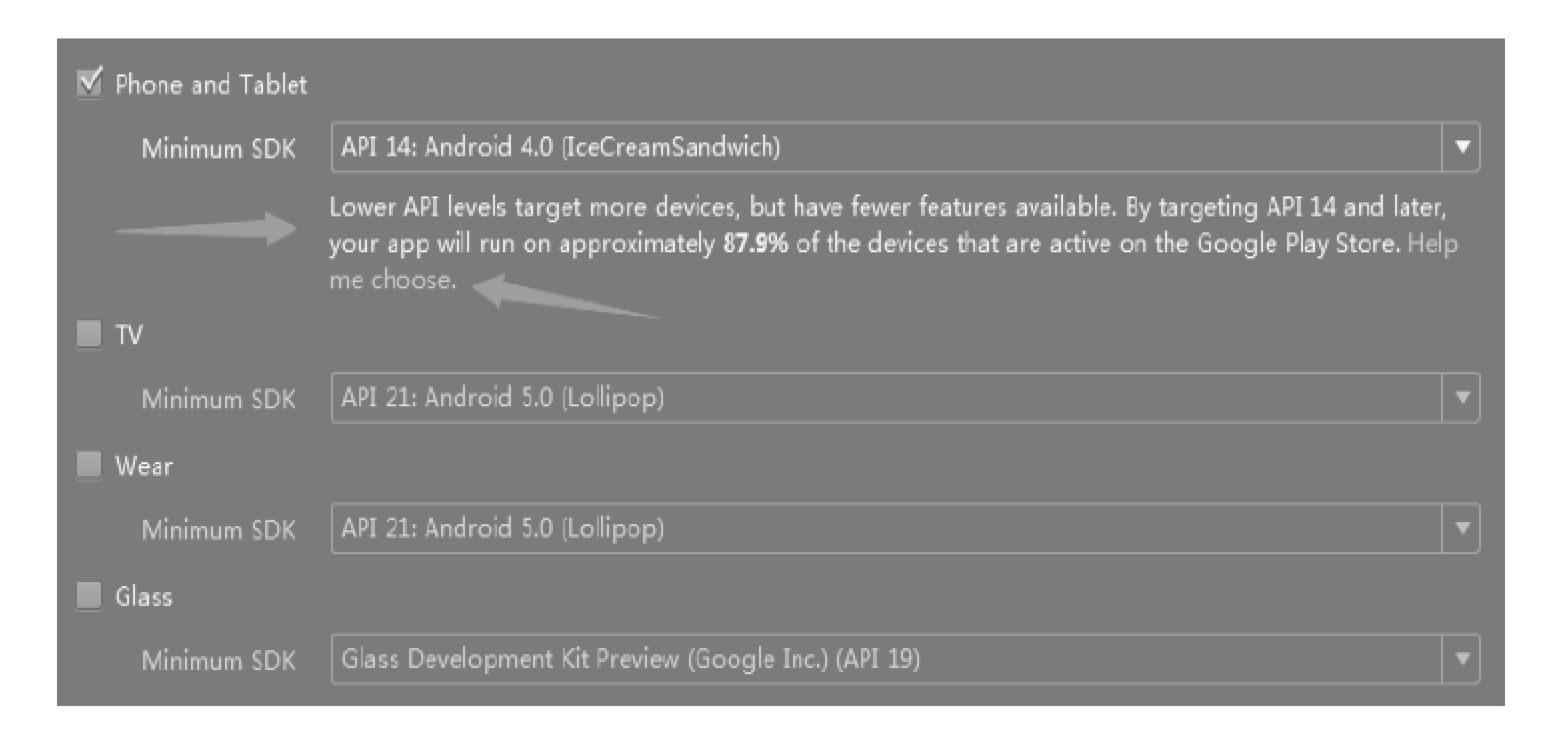


图 1-18　选择适配的设备

在新建 App 选择最低适配版本时，强大的 Android Studio 会给出一些有用的版本统计提示，单击 Help me choose 后弹出更加形象的分布图表描述，以帮助用户选择，如图 1-19 所示。

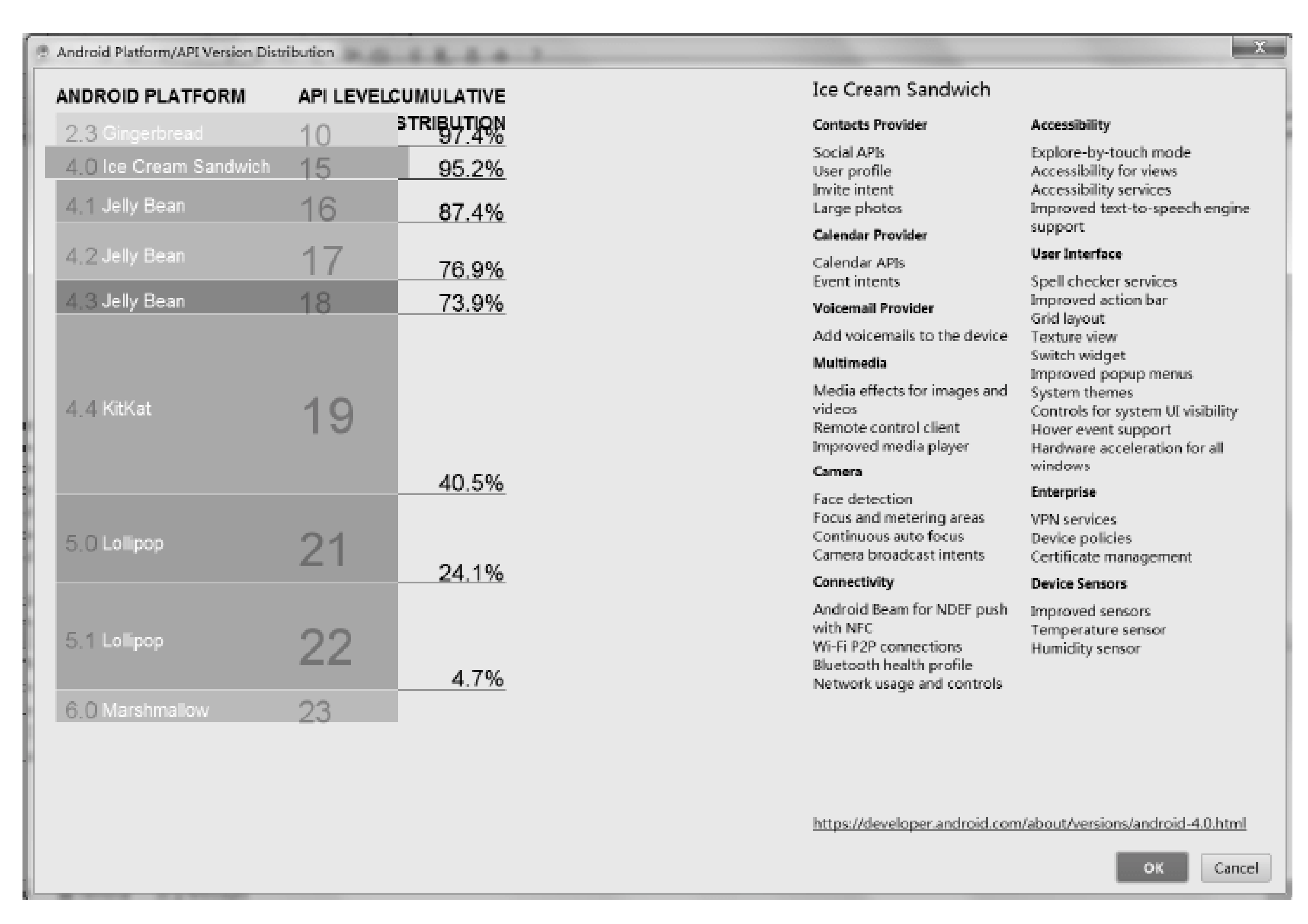


图 1-19　Android studio 中版本统计提示

当选择完 App 要适配的设备以及版本支持之后会进入选择 Activity 类型的界面，如图 1-20 所示。这里我们选择一个 Empty Activity。

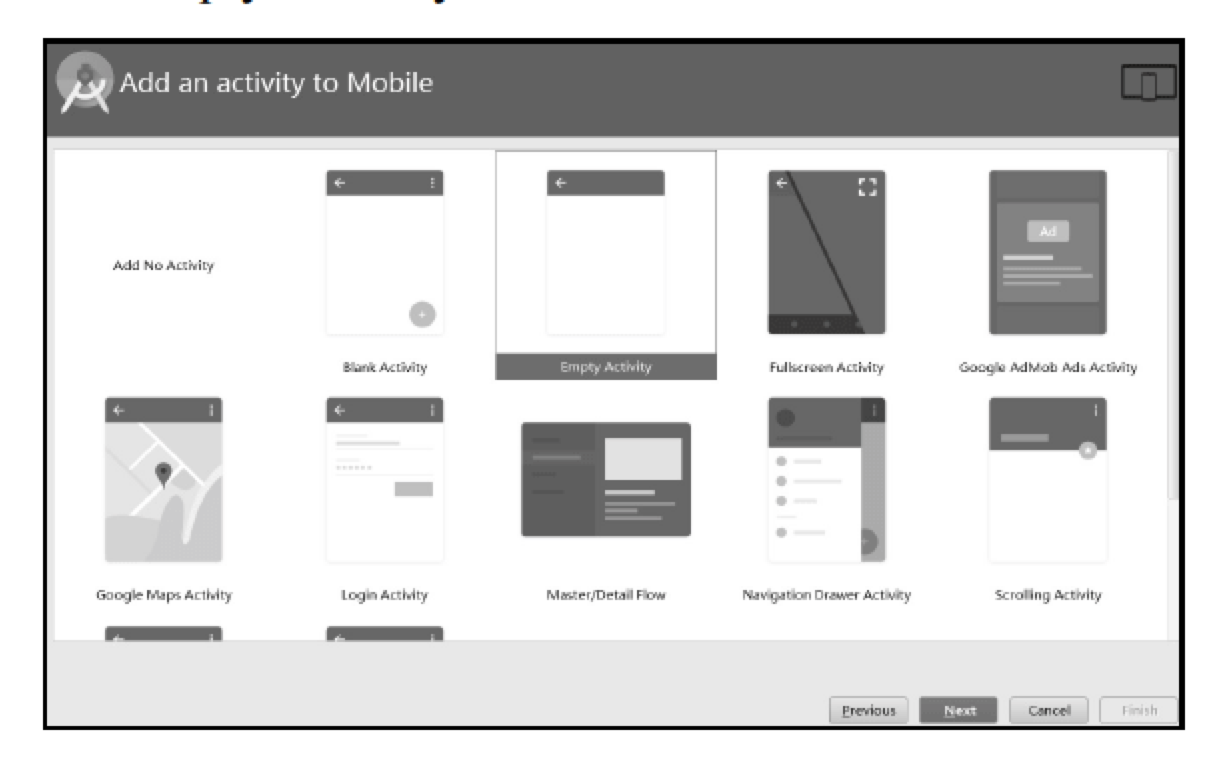


图 1-20　选择 Activity

单击 Next 按钮就会进入设置 Activity 名称的界面，如图 1-21 所示。这个名称可以根据需要随意设置。

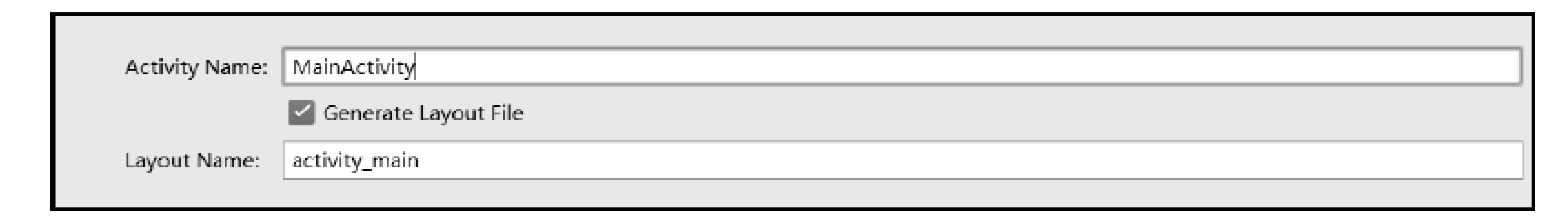


图 1-21　设置 Activity 的名称

设置完成后就可以进入工程界面了。第一次安装工程初始化时需要联网下载 Gradle，速度会比较慢，有时不是第一次安装也会慢，因为工程依赖的 Gradle 版本不匹配时也会自动重新下载。在等待一段时间之后，会进入如图 1-22 所示的工程界面。

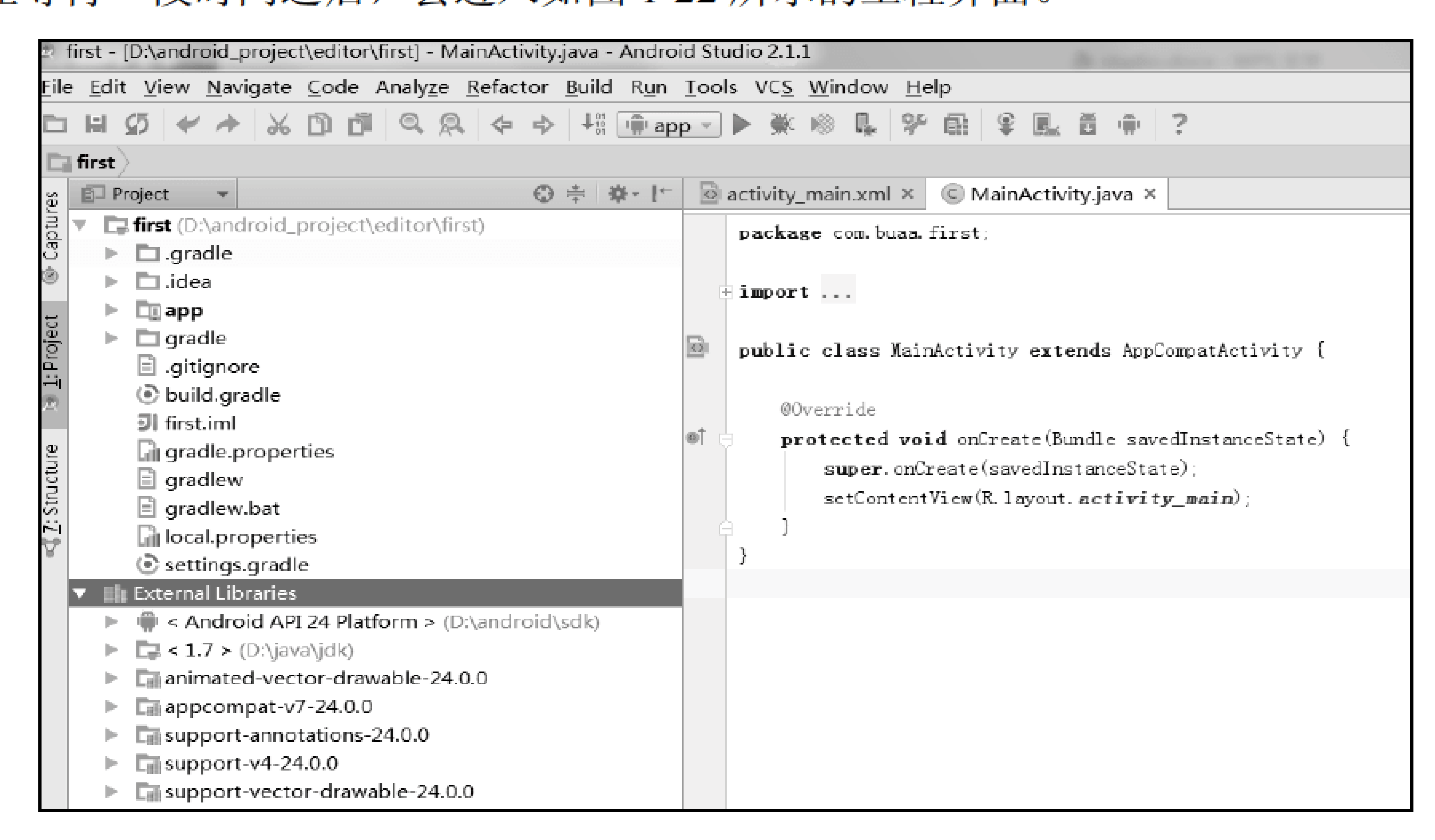


图 1-22　新建工程界面

到此，一个使用 Android Studio 建立的 Android 工程就完成了。连接真机或者打开模拟器，单击 app ▶ 上面的红色三角就可以运行这个 Android 应用了。

1.4.2　创建模拟器并使用模拟器运行应用

Android 模拟器是可以运行在计算机上的虚拟设备，无须使用物理设备即可预览、开发和测试 Android 应用程序。当你身边并没有合适的 Android 设备时，模拟器就是一个不错的选择。

在 Android Studio 主界面上方的工具栏中有一个名为 AVD Manager 的按钮，单击它就能打开 Android 虚拟设备管理器（Android Virtual Device，AVD）。第一次使用时并没有任何的虚拟设备，我们需要单击中央的 Create a virtual device 按钮来创建一台模拟器，如图 1-23 所示。

创建模拟器的第一步是选择硬件。你可以通过选择现有的设备模板来定义一台模拟器。在图 1-24 所示左侧的 Category 分类中可以选择要创建哪种类型的设备，通常是开发手机上的应用，所以选择 Phone 就可以了；右侧则显示了所有 Google 官方的设备模板，比如历年来发布的 Nexus 系列以及 Google Phone 系列。需要注意的是，此处只是选择型号对应的硬件条件，而不会选择该设备在发布时搭载的系统镜像。

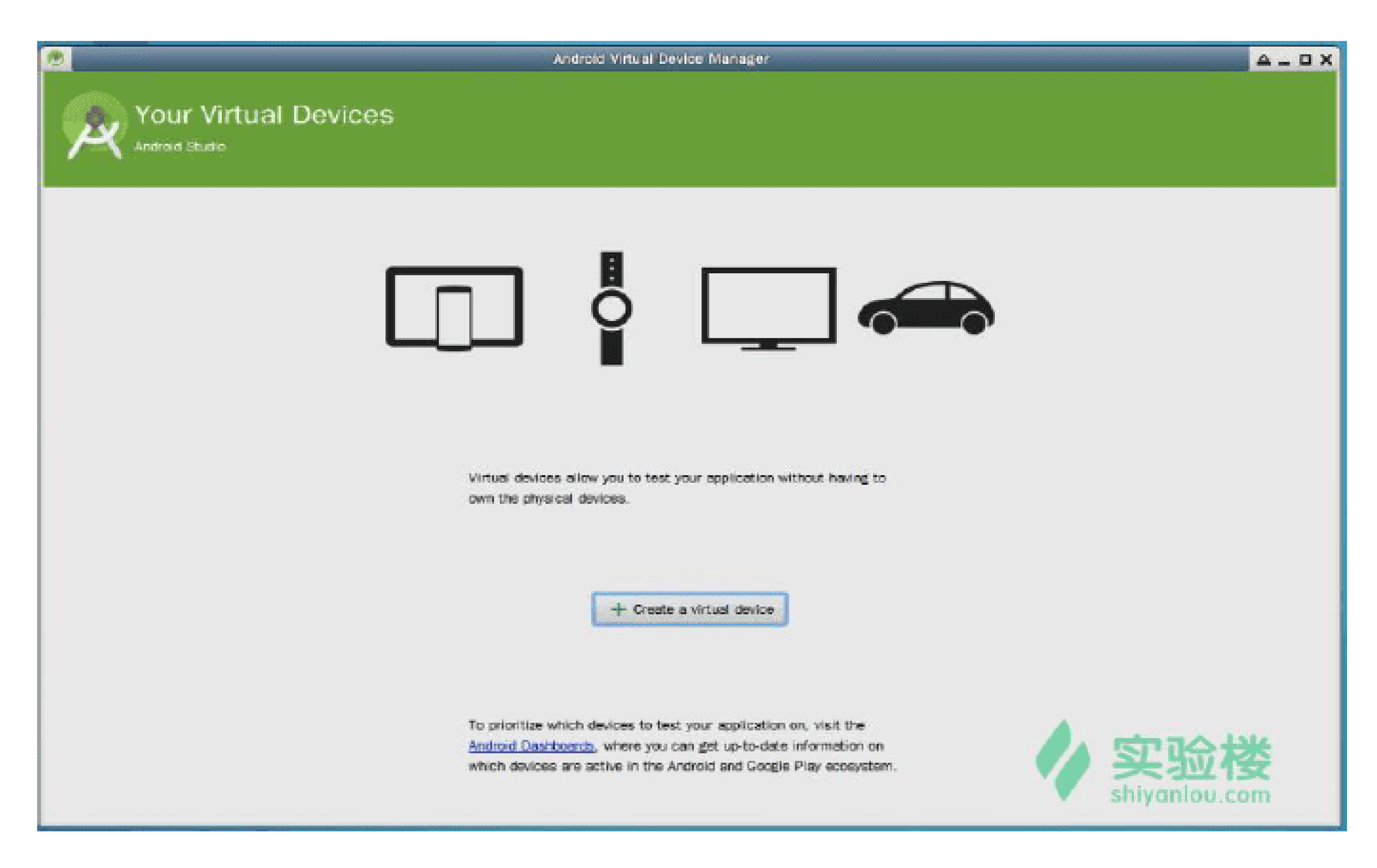


图 1-23　新建一个模拟器

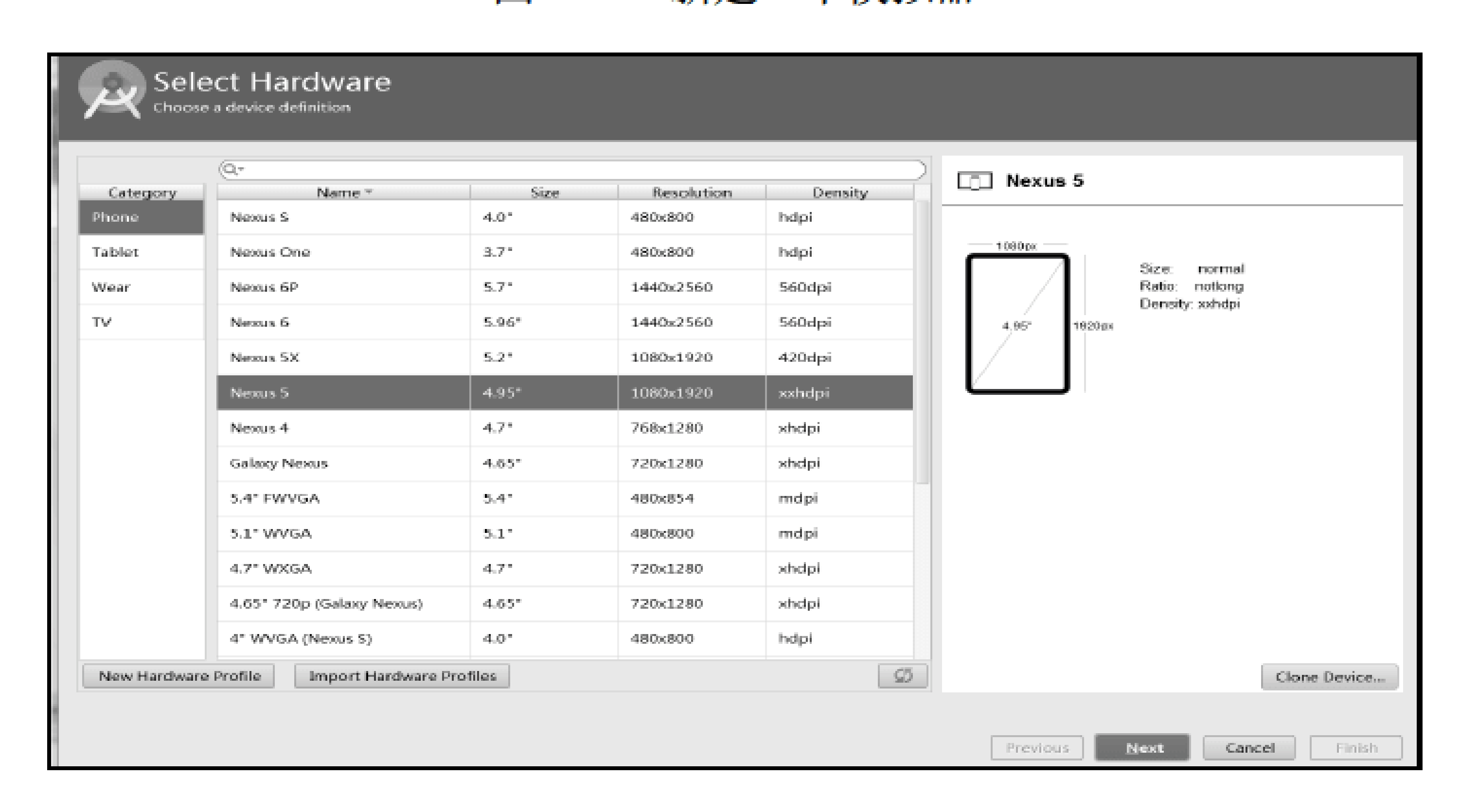


图 1-24　选择硬件

也就是说，你可以单击左下角的 New Hardware Profile 按钮定义一台设备的硬件配置和外观，或者通过 Import Hardware Profiles 按钮来导入现成的配置方案。

单击右下角的 Next 按钮，进入系统镜像选择界面，如图 1-25 所示。

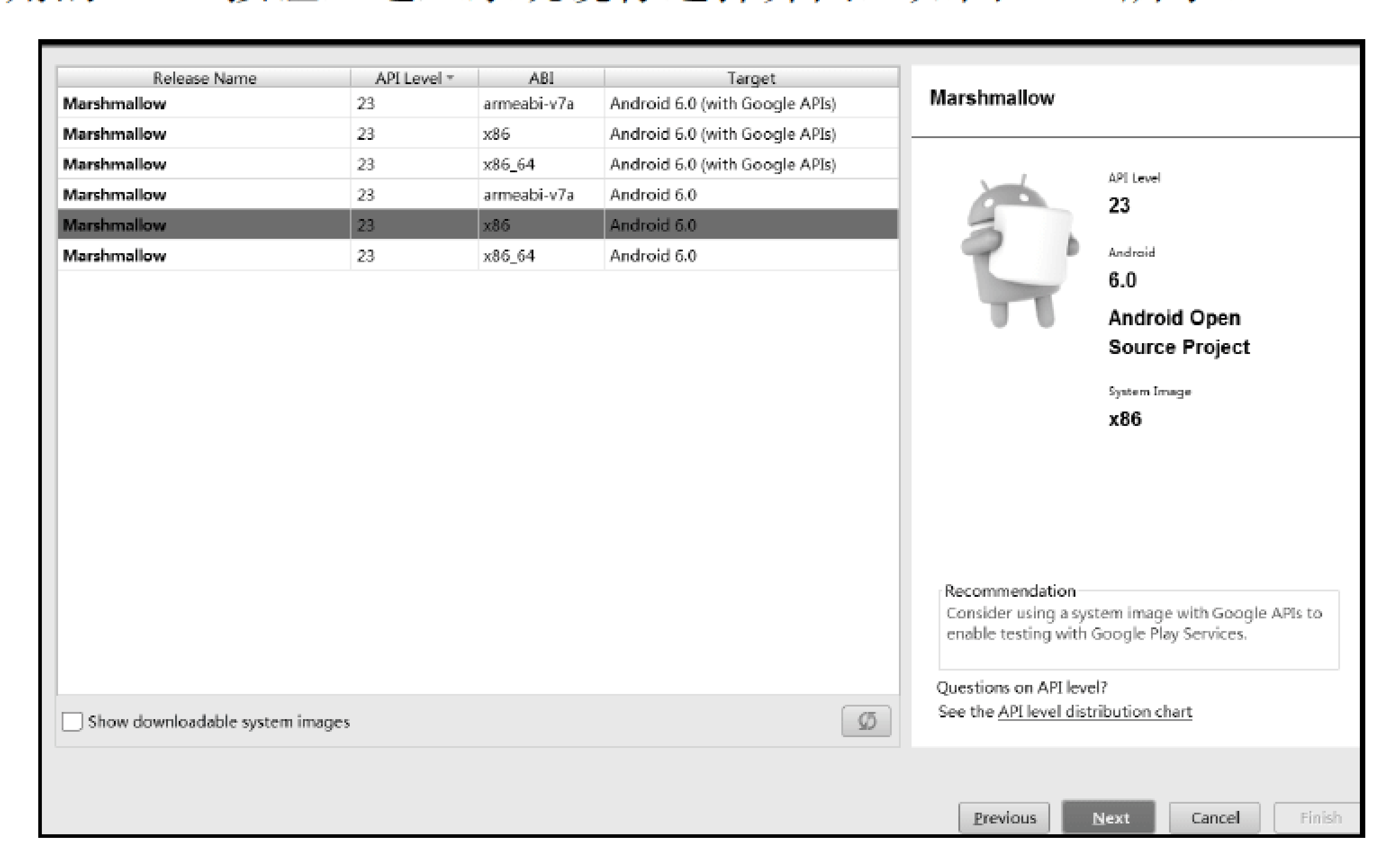


图 1-25　选择系统镜像

我们常说某个 Android 手机是 5.0 或 6.0 的系统，这里的 5.0 或 6.0 就是指系统镜像的版本。同样，对于模拟器而言，也需要为其配置某个版本的系统镜像。你可以看到这里有 6 个镜像可供选择，这里选择第五项 Android 6.0 版本支持 x86 的镜像，据官方文档报道此镜像的模拟器速度较快。

如果需要其他版本的系统，可以在 Android SDK Manager 中下载对应的系统镜像包，再进入 AVD Manager 就能看到它们了。

接着，单击右下角的 Next 按钮，进入确认配置界面，如图 1-26 所示。

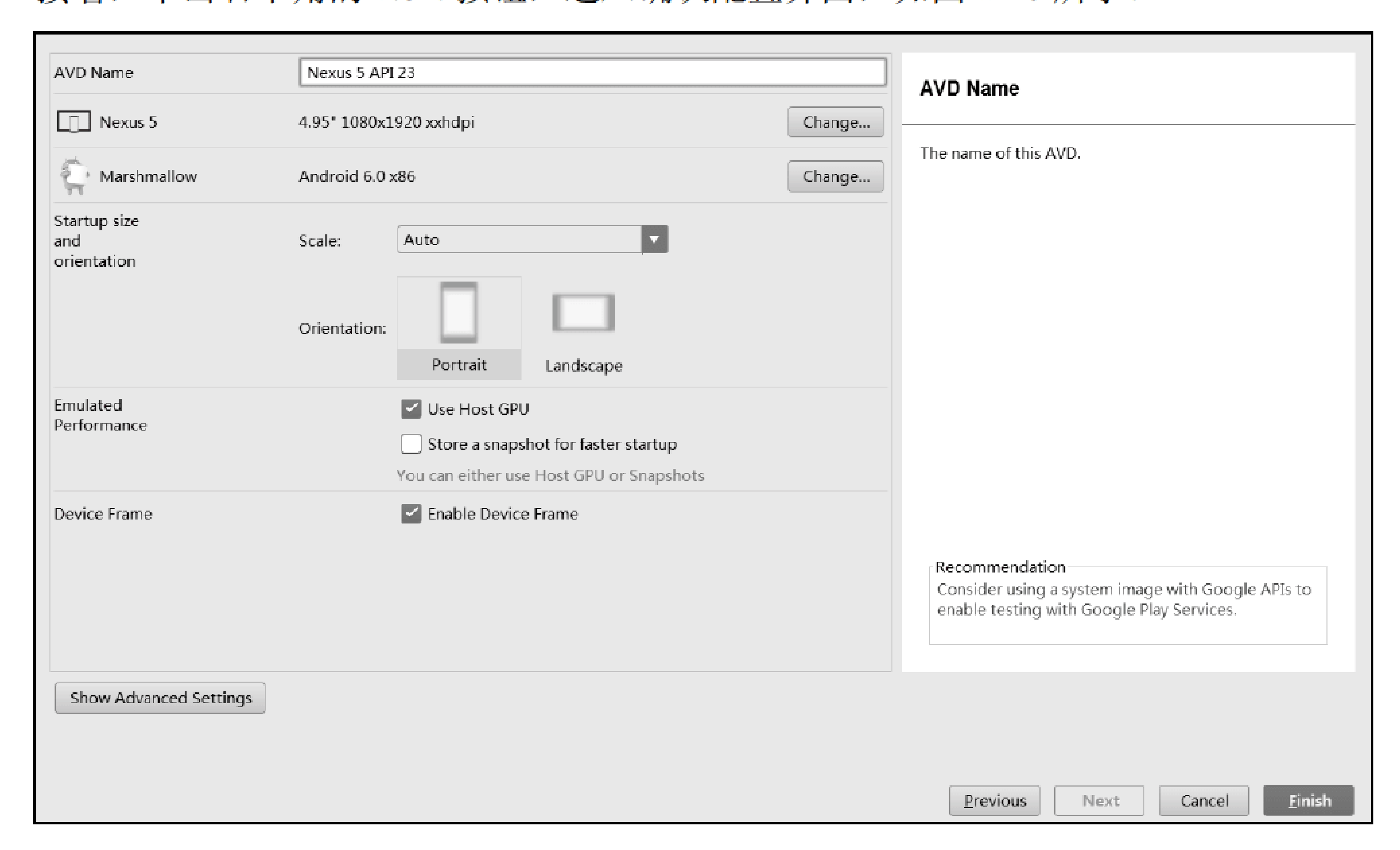


图 1-26　确认配置界面

在这里，可以设置模拟器的名称。其他选项无须特别设置。在实际的开发工作中，建议通过 USB 数据线将运行着 Android 系统的设备（手机或平板）与电脑相连接。这样便能在较高性能的设备上测试应用，而不是体会模拟器带来的卡顿感。

最后单击Finish按钮就能在AVD Manager的列表中看到刚刚创建的模拟器，如图 1-27 所示。

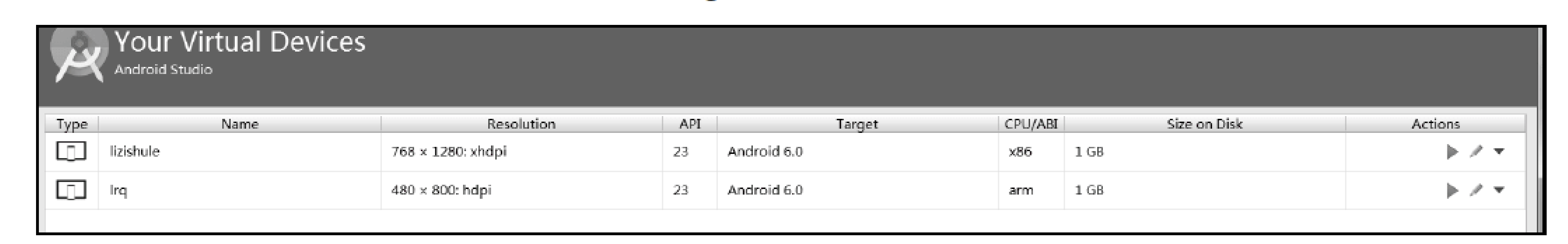


图 1-27　刚创建的模拟器

单击启动按钮打开模拟器，会看到如图 1-28 所示的模拟器。

接着单击 Android Studio 工程上方 app 中的红色三角按钮启动应用，如图 1-29 所示。



图 1-28 模拟器

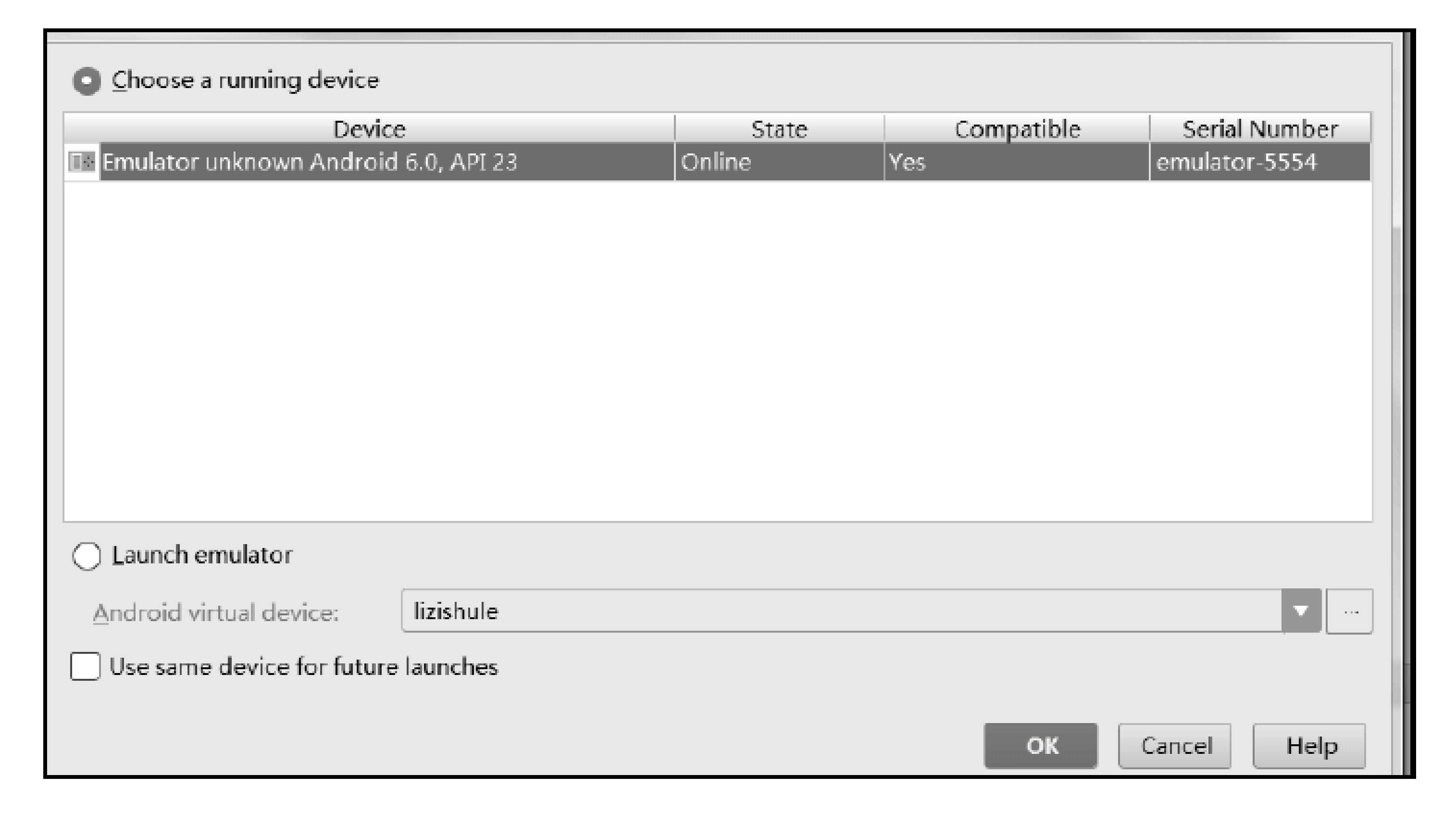


图 1-29 启动应用

选择使用模拟器打开，单击 OK 按钮。第一个 Android 应用就成功运行了，如图 1-30 所示。



图 1-30 成功运行的第一个应用

1.4.3 工程目录分析

新建的 App 的整体目录结构如图 1-31 所示。

工程中的文件可大致分成 3 块，即编译系统（Gradle）、配置文件、应用模块（app）。Gradle 是 Google 推荐使用的一套基于 Groovy 的编译系统脚本，图 1-31 中出现 gradle 字眼的就是 gradle 相关的一些文件。除了 app 文件夹以外，大部分都是配置文件，它们的功能如表 1-2 所示。

表 1-2 中是与外部文件相关的一些文件介绍，更重要的 app 模块里的文件目录结构如图 1-32 所示。

表 1-3 列出了 app 目录中文件及文件夹的用途。

表1-2　配置文件的名称与功能

文件（夹）名	用途
.gradle	Gradle 编译系统，版本由 wrapper 指定
.idea	Android Studio IDE 所需要的文件
build	代码编译后生成的文件存放的位置
gradle	wrapper 的 jar 和配置文件所在的位置
.gitignore	git 使用的 ignore 文件
build.gradle	gradle 编译的相关配置文件（相当于 Makefile）
gradle.properties	gradle 相关的全局属性设置
gradlew	*nix 下的 gradle wrapper 可执行文件
graldew.bat	Windows 下的 gradle wrapper 可执行文件
local.properties	本地属性设置（key 设置，Android sdk 位置等属性），这个文件是不推荐上传到 VCS 中去的
settings.gradle	和设置相关的 gradle 脚本

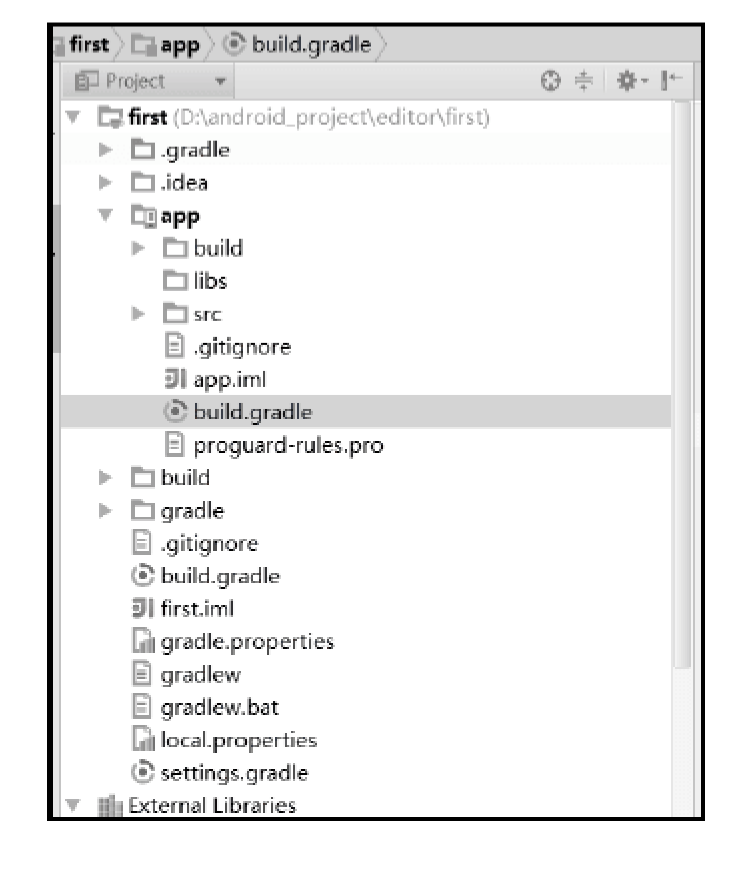




图 1-31　新建工程的目录结构

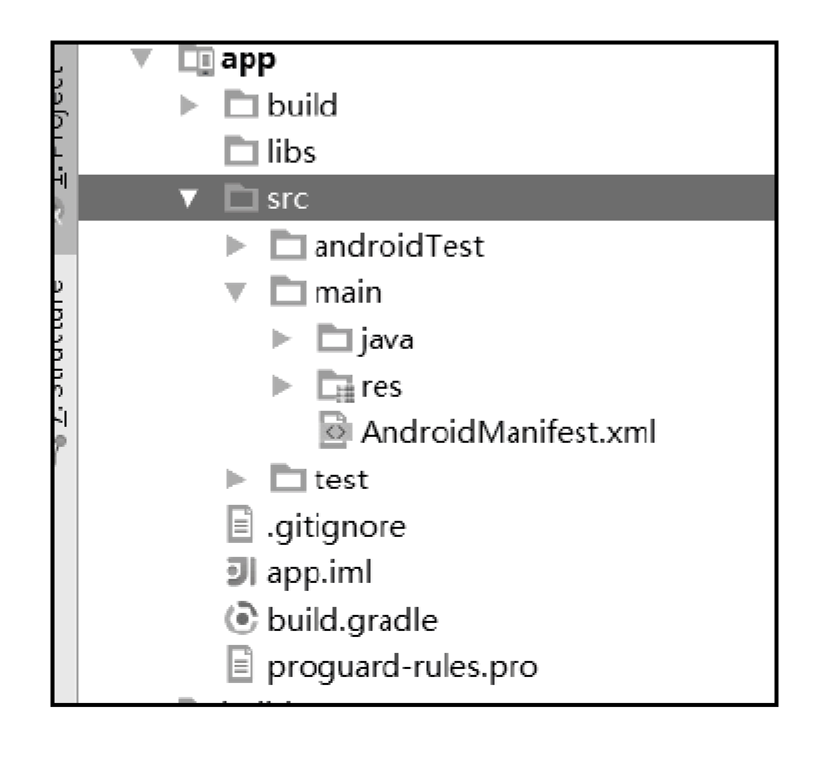


图 1-32　app 模块文件目录结构

表1-3　app模块文件目录结构说明

文件（夹）名	用途
build	编译后的文件存在的位置（包括最终生成的 apk）
libs	依赖的库所在的位置（jar 和 aar）
src	源代码所在的目录
src/main	主要代码所在位置（src/androidTest 就是测试代码所在位置了）
src/main/assets	Android 中附带的一些文件
src/main/java	java 代码所在的位置
src/main/jniLibs	jni 的一些动态库所在的默认位置（.so 文件，本项目中没有使用，但是它是存在的）
src/main/res	Android 资源文件所在的位置
src/main/AndroidManifest.xml	Android 工程的清单文件

（续表）

文件（夹）名	用途
build.gradle	和这个项目有关的 gradle 配置，相当于这个项目的 makefile，一些项目的依赖就在这里面
Proguard-vules.pro	代码混淆配置文件

1.4.4 Android Studio 常见问题

第一次使用 Android Studio 可能会遇到一些问题，根据笔者的开发经验，新手在使用 Android Studio 时，常会被下面一些问题困扰。

1. 中文乱码问题

在开发中，有时会遇到中文乱码问题，要解决它，只需要在窗口中找到 IDE Settings→Appearance，在右侧勾选上 Override default fonts by，然后在第一个下拉框中选择字体 simsun，然后单击 apply，重启 IDE。

2. 如何设置快捷键

在 settings 窗口中找到 IDE Settings→keymap，右侧打开的就是快捷键了。右击要修改的快捷键会弹出一个菜单，选择 Add keyboard shortcut 就可以修改快捷键了。若要删除，则在弹出的菜单中选择 remove XXX。



在 Android Studio 的快捷键设置里可以直接设置使用 Eclipse 快捷键或其他 IDE 快捷键。如果你热衷 Eclipse，也可设置成 Eclipse 的快捷键。

3. 如何将 Eclipse 工程导入 Android Studio 使用

选择 File→Import Project，在弹出的菜单中选择要导入的工程，然后直接单击 Next 按钮，在第二个窗口中选择默认的第一个选项即可。需要注意的是，在 Android Studio 中有两种工程，一种是 Project，一种是 Module。

4. 导入 jar 包问题

选择 File→Project Structure，在弹出的窗口左侧找到 Libraries 并选中，然后单击+按钮，并选择 Java 就能导入 Jar 包了。或者直接复制 jar 文件到项目的 libs 文件夹下，然后运行 Sync Project with Gradle Files，再 clean project 重新编译。当然，Android Studio 支持 Gradle，所以我们也可以直接在 Gradle 配置文件中加入 jar 包的链接，让 Gradle 帮助加载 jar 包。

5. 如何删除项目

Android Studio 对工程删除做了保护机制，默认在项目右键里没有删除选项，并且 module 上面有一个小手机。删除的第一步就是去掉保护机制，也就是让小手机不见，具体做法是在工程上右击，选择 open module setting，或者按 F4 键进入设置界面，选中所要删除的 module，然后单击减号，取消保护机制，项目工程右键就有删除选项了。注意：删除时会将源文件删除。

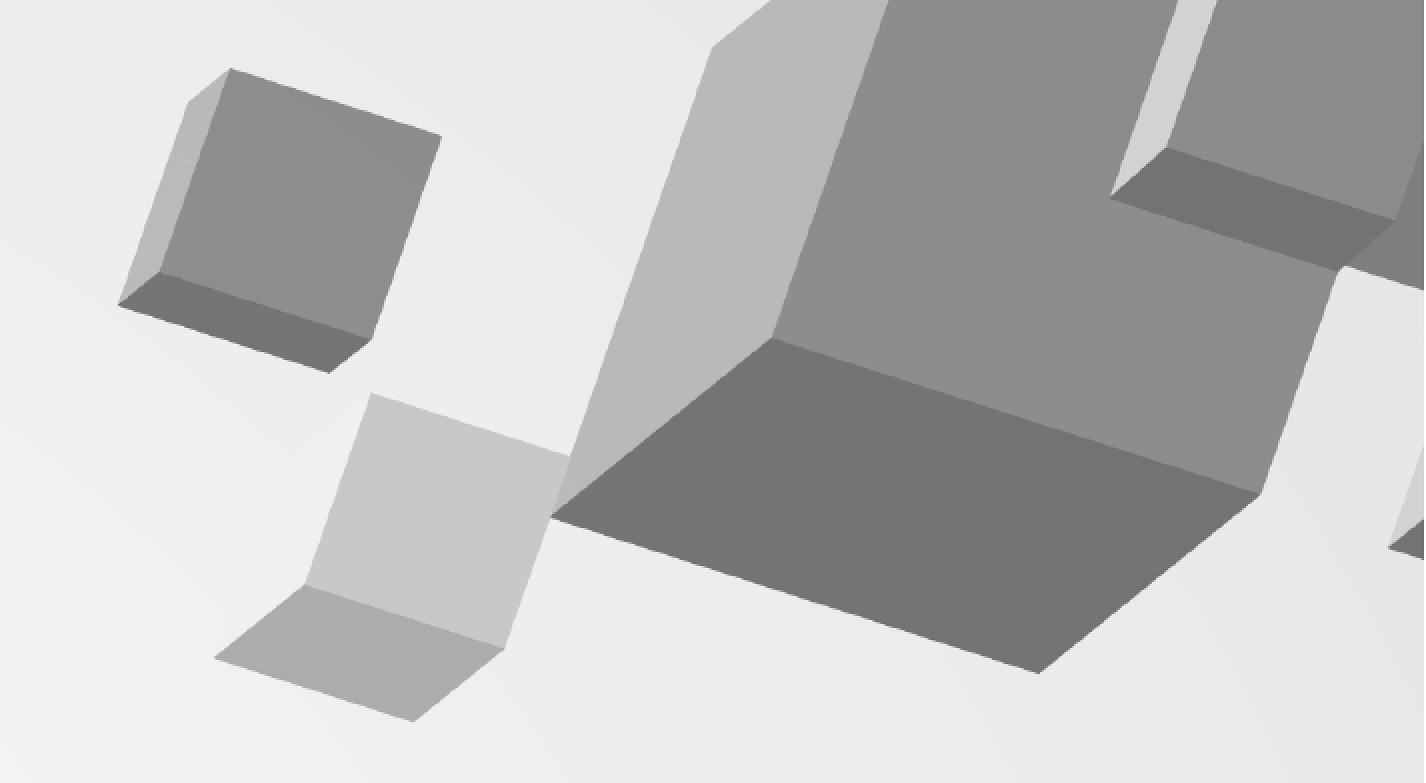
6. 如何修改主题

在 IDE Settings→Appearance 右侧的 Theme 中选择自己喜欢的主题即可。

1.5 小　结

本章充分阐述了 Android 系统的相关认识，并且演示了如何搭建 Android 开发环境，并创建了第一个 Android 项目，同时对 Android 项目的目录结构进行了分析。（本章讲解的 Android 开发都是基于 Android Studio 进行的。）

对于项目目录结构，如果不能完全理解也很正常，毕竟里面有太多的内容都没有接触过。不用担心，这并不会影响到后面的学习。等到学完整本书后再回来看这个目录结构图，就会觉得特别清晰、简单了。



第 2 章 界面组件 Activity

在第 1 章我们讲解了如何基于 Android studio 搭建 Android 开发环境，并建立了第一个 Android 工程，分析了 Android 工程的目录结构。读者对 Android 开发有了一个大致的了解，本章将讲解 Android 开发中最重要的组件 Activity——Android 应用程序的界面，凡是在应用中能够看到的东西都是放在 Activity 中的。

2.1 从第一个工程开始

使用 Android 手机的读者都会使用手机中的 App，但是这些 App 是如何运行的呢？可能大部分人都不知道。本节将以第 1 章创建的工程为例来讲解一个 App 是如何运行的，并介绍一个 Android 工程中的资源文件有哪些，同时对 Activity 进行简单的探究。

2.1.1 App 是如何运行的

本节我们将一起分析一个 App 究竟是怎么运行起来的，这对于开发 App 很有帮助。以第 1 章所创建的工程为例，打开 AndroidManifest.xml 文件，从中找到如下代码：

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.buaa.first">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

这段代码表示对 MainActivity 这个 Activity 进行注册。没有在 AndroidManifest.xml 里注册的 Activity 是不能使用的。其中，intent-filter 里的两行代码非常重要，<action android:name="android.intent.action.MAIN" />和<category android:name="android.intent.category.LAUNCHER" />表示 MainActivity 是这个项目的主 Activity，在手机上点击应用图标，首先启动的就是这个 Activity。

那 MainActivity 具体又有什么作用呢？第 1 章中使用模拟机运行 App 的效果图中显示的其实就是 MainActivity 这个 Activity。代码如下：

```
package com.buaa.first;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

首先，MainActivity 继承自 AppCompatActivity 类，而 AppCompatActivity 类继承自 Activity 类。Activity 是 Android 系统提供的一个 Activity 基类，Android 项目中所有的 Activity 都必须继承它才能拥有 Activity 的特性。然后，MainActivity 中有一个 onCreate()方法。onCreate()方法是一个 Activity 被创建时必定要执行的方法，其中只有两行代码，并且没有 Hello world!字样。第 1 章效果图中显示的 Hello world!是在哪里定义的呢？

其实 Android 程序的设计讲究逻辑和视图分离，因此是不推荐在 Activity 中直接编写界面的，更加通用的一种做法是先在布局文件中编写界面，然后在 Activity 中引入。在 onCreate()方法的第二行调用了 setContentView()方法，就是这个方法给当前的 Activity 引入了一个 activity_main.xml 布局，Hello world!就是在这里定义的。布局文件都是定义在 res/layout 目录下的，当展开 layout 目录时，会看到 activity_main.xml 这个文件。打开之后的代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="com.buaa.first.MainActivity">

   <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/hello_world" />
</RelativeLayout>
```

后面会对布局进行详细讲解，现在只需要知道上面的代码中有一个 TextView，这是

Android 系统提供的一个控件，是用于在布局中显示文字的即可。在 TextView 中还有 hello world 的字样，但是这并不是在效果显示图中显示的“Hello World！”。细心的读者会发现感叹号没了，大小写也不一样。

真正的“Hello world!”字符串也不是在布局文件中定义的。Android 不推荐在程序中对字符串进行硬编码，更好的做法是先把字符串定义在 res/values/strings.xml 里，然后在布局文件或代码中引用。打开 strings.xml，里面的内容如下：

```
<resources>
    <string name="app_name">first</string>
    <string name="hello_world">Hello World!</string>
</resources>
```

“Hello world!”字符串就是定义在这个文件里的，并且字符串的定义都采用键值对的形式，Hello world!值对应了一个叫作 hello_world 的键，因此在 activity_main.xml 布局文件中通过引用 hello_world 这个键就能找到相应的值。

activity_main.xml 布局文件中还有一个叫作 app_name 的键，可以通过修改 app_name 对应的值来改变应用程序的名称。那么到底是哪里引用了 app_name 这个键呢？在 AndroidManifest.xml 文件中有答案。读者可以自行研究。

2.1.2 项目中的资源

展开 res 目录，里面的内容很多，很容易让人看得眼花缭乱，如图 2-1 所示。

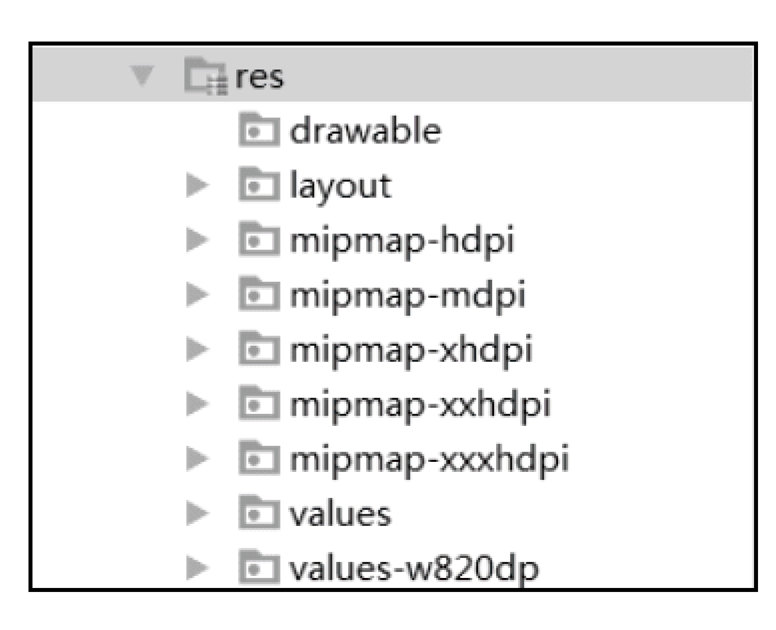


图 2-1 res 目录下的资源

简单来说，图 2-1 中的 drawable 文件夹是用来放图片的，所有以 values 开头的文件夹都是用来放字符串的，layout 文件夹是用来放布局文件的，以 mipmap 开头的文件夹是用来放置应用图标的。

在开发过程中，引用这些资源有两种方式，以刚刚在 strings.xml 中找到的“Hello world!”字符串为例：

- 在代码中通过 R.string.hello_world 可以获得该字符串的引用。
- 在 XML 中通过@string/hello_world 可以获得该字符串的引用。

基本的语法就是上面两种方式。对于第二种引用方式，读者经过上面的分析应该可以理解，而对于第一种方式就可能不太了解了。其中，R 指的是 Android 中的 R 类。R 类是将 Android 的资源文件存储为键值对的一个类，会将资源文件按照不同类型建立静态内部类，在内部类内部用键值对来存储。第一种引用方式就是在代码中引用 R 类下的静态内部类 string 类内键为“hello_world”的字符串。本应用中的 Activity 对 R.layout.activity_main 的调用就是同样的道理。特别提醒一下读者，此处的 R 类是系统自行生产并维护的，请勿修改，Android Studio 为了避免开发者修改 R 类文件，特意把 R 类文件移出了。

res 文件中的这些资源文件都是可以替换的。如果想要修改字符串，只需要到 values 文件夹下的 strings.xml 文件中修改键值对的值。如果想要修改图片资源只要修改 drawable 和

mipmap 文件夹中的图片就可以。修改布局也是同样道理，修改 layout 文件夹下的布局文件就可以了。first 项目的图标就是在 AndroidManifest.xml 中通过 android:icon="@drawable/ic_launcher"来指定的，ic_launcher 图片在以 mipmap 开头的各个文件夹下。如果想要更换图标该怎么做呢？读者可以试一下。

2.1.3　理解 Activity

通过前面两个部分对 Android 工程的分析，读者现在应该对一个 Android 应用的整体结构有了更加深入的了解，也会发现在一个 Android 工程中 Activity 处在一个极其重要的位置。Android 应用的界面就是通过 Activity 调用 res 中的资源显示出来的。

Activity 是 Android 组件中最基本也是最为常见用的四大组件（Activity、Service、Content Provider、BroadcastReceiver）之一。

一个 Activity 是一个应用程序组件，提供一个屏幕，可以供用户为了完成某项任务而进行交互，例如拨号、拍照、发送 email、看地图。每一个 Activity 都被给予一个窗口，在上面可以绘制用户接口。窗口通常充满屏幕，也可以小于屏幕而浮于其他窗口之上。

一个应用程序通常由多个 Activit 组成，它们通常是松耦合关系。通常，一个应用程序中被指定为"main"activity 的 Activity 是第一次启动应用程序的时候呈现给用户的那个 Activity。每一个 Activity 都可以启动另一个 Activity 来完成不同的动作。每一次一个 Activity 启动，前一个 Activity 就停止了，但是系统保留 Activity 在一个栈上（Back Stack）。当一个新 Activity 启动时，它会被推送到栈顶，取得用户焦点。Back Stack 符合简单"后进先出"原则，所以当用户完成当前 Activity 后单击 Back 按钮，它会被弹出栈（并且被摧毁），然后之前的 Activity 恢复。

当一个 Activity 因新的 Activity 启动而停止时，会通过 Activity 的生命周期回调函数通知这种状态转变。一个 Activity 可能会收到许多回调函数，这源于它自己的状态变化——无论系统创建它、停止它、恢复它、摧毁它——并且每个回调提供完成适合这个状态指定工作的机会。例如，当停止的时候，Activity 应该释放任何大的对象，例如网络数据库连接。当 Activity 恢复时，可以重新获得必要的资源和恢复被中断的动作。这些状态转换都是 Activity 的生命周期部分（2.2 节会详细论述）。

创建一个 Activity，必须创建一个 Activity 的子类（或者一个 Activity 子类的子类），必须实现 onCreate()方法。当创建 Activity 的时候系统会调用它。在我们的实现中，应该初始化 Activity 的基本组件。更重要的是，这里是我们必须调用 setContentView（）来定义 Activity 用户接口的地方。

Android 提供大量预定义的 View，用于设计和组建布局。Widget 是一种给屏幕提供可视化（并且交互）元素的 View，例如按钮、文件域、复选框或者仅仅是图像。Layouts 是继承于 ViewGroup 的 View，为子 View 提供特殊的布局模型，例如线程布局、格子布局或相关性布局。我们可以子类化 View 和 ViewGroup 类（或者存在的子类）来创建自己的 Widget 并应用到 Activity 布局中。

最普通的方法是使用 View 定义一个布局，一起和 XML 布局文件保存在程序资源里。这样就可以单独维护我们的用户接口设计（与定义 Activity 行为的代码无关）。可以使用

setContentView()设置布局 UI，传递资源布局的资源 ID。

同时，必须在 manifest 文件中声明 Activity，否则将不能被系统访问。声明格式如下：

```
<application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:supportsRtl="true"
    android:theme="@style/AppTheme">
    <activity android:name=".MainActivity">
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />

            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
</application>
```

一个<activity>元素能指定多种 intent filters——使用<intent-filter>元素——声明其他应用程序时可以将其激活。当使用 Android SDK 工具创建一个新应用程序时，会自动创建存根 Activity，包含一个 intent filter，声明 activity 响应“main”动作，并且被放置在“launcher”分类中。

<action>元素指定这是一个“main”入口点。<category>元素指定这个 Activity 应该被列入系统应用程序列表中（为了允许用户启动这个 Activity）。

如果希望应用程序自包含并且不希望其他应用程序激活它的 activities，那么不需要任何 intent filters。只有一个 Activity 应该有“main”动作和“launcher”分类，就像前面这个例子。不希望被其他应用程序访问时原 Activities 应该没有 intent filters。

如果我们希望 Activity 从其他应用程序响应隐含的 intents，就必须为这个 Activity 定义额外的 intent filters。每一种你希望响应类型的 intent 都必须包含<intent-filter>、<action>元素，可选的有<category>元素或<data>元素。这些元素指定 Activity 能响应的 intent 的类型。

除了上面简述的 Activity 的生命周期，以及如何创建一个 Activity、如何声明这个 Activity 之外，Activity 还有一些常用的基本方法，也比较重要，如表 2-1 所示。

表2-1　Activity的常用方法

方法名	方法描述
public final View findlliewById(int id)	根据组件 id 获取组件对象
public void setEnabled(boolean enabled)	设置是否可编辑
public Window getWindow()	获取一个 Window 对象
public void setContentView(int layoutResID)	设置显示组件
public void setContentView(View view)	设置显示组件
public void addContentView(View view)	动态添加组件

表 2-1 中只列出了 Activity 的基本方法，除此之外，Activity 类还提供了与 intent、service 等相关的方法，等讲解相关知识点时再做阐述。

2.2 Activity 生命周期

每一种技术都有其生命周期，例如 Java EE 技术中的 Servlet 生命周期就分为 4 步，即实例化、初始化、服务和销毁。当然 Android 系统中的 Activity 也不例外，也有其自己的生命周期过程。本节将讲解 Activity 的生命周期，并通过实例演示的方法带领读者仔细观察在 Activity 的生命周期中到底发生了什么。

2.2.1 Activity 生命周期概述

Activity 是由 Activity 栈进行管理的。当来到一个新的 Activity 后，此 Activity 将被加入 Activity 栈顶，之前的 Activity 位于此 Activity 底部。Activity 一般有 4 种状态：

- 当 Activity 位于栈顶时，正好处于屏幕最前方，处于运行状态。
- 当 Activity 失去焦点但仍然对用户可见（如栈顶的 Activity 是透明的或者栈顶 Activity 并不是铺满整个手机屏幕）时，处于暂停状态。
- 当 Activity 完全被其他 Activity 遮挡时， Activity 对用户不可见，处于停止状态。
- 当 Activity 由于人为或系统原因（如低内存等）被销毁时，处于销毁状态。

在每个不同的状态阶段，Android 系统都对 Activity 内相应的方法进行了回调。在开发过程中写的 Activity 一般都继承 Activity 类并重写相应的回调方法。Google 官方提供的 Android 中典型的生命周期流程图如图 2-2 所示。读者可以通过观察这个图获得一个大致的印象。此图对于理解 Activity 生命周期很重要，读者可以多做观察，熟记于心。

通过这个 Activity 流程图，我们可以看出 Activity 生命周期中的 7 个事件。

- onCreate(): 当 Activity 第一次被创建时调用，可以在这个方法中绑定数据或创建其他视图控件，其中应该注意的问题是，覆写 onCreate()方法时尽量将当前的 Activity 状态保存进系统，以备以后再使用这个 Activity 时保存以前界面的状态。
- onStart(): 当 Activity 变为用户可见之前调用。
- onResume(): 当 Activity 可以与用户交互之前调用，也就是 Activity 对象到达 Activity 栈的顶部即将成为前台进程时被调用。
- onPause(): 当系统调用其他 Activity 对象时调用，可以在这个方法中将当前 Activity 对象没有保存的数据保存到持久化对象中，也可以在这个方法中结束比较耗费 CPU 时间的操作，比如动画之类的。用这个方法写的代码要尽量效率高一些，如果这个方法没有执行完，新的 Activity 对象将不会显示出来，因为会影响客户的体验性。也就是说，新的 Activity 对象必须等待 onPause()方法执行完毕后再显示出来。大多数情况下，在 onPause()方法中要关闭 onResume()中打开的资源。

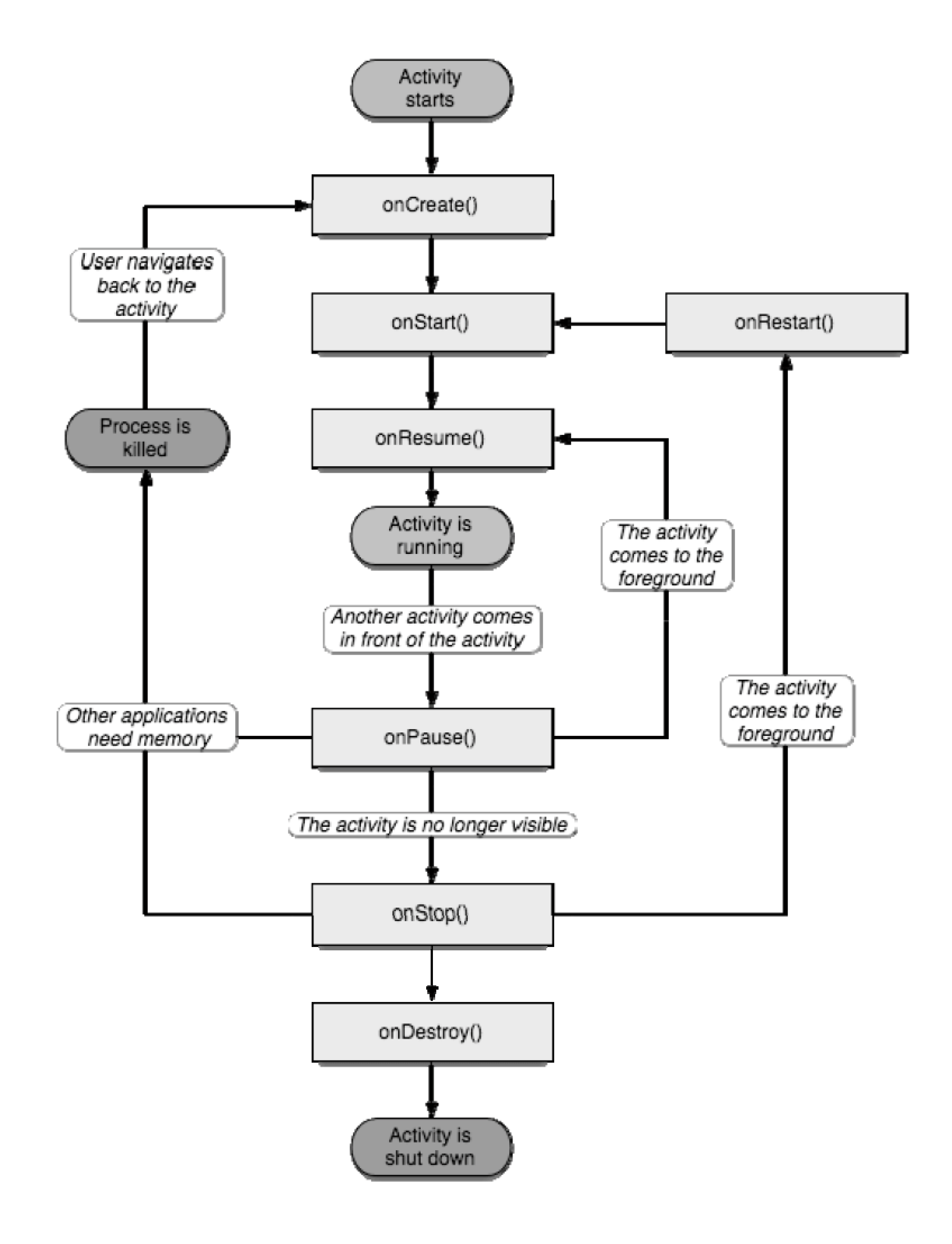


图 2-2 Activity 生命周期流程图

- onStop(): 当 Activity 不可视时调用。
- onDestroy(): 当销毁 Activity 对象时调用。
- onRestart(): 当处于 onStop()状态的 Activity 又变为可视时调用。

除了上述的 7 个生命周期时间之外，还有 3 个关键的周期循环：

- Activity 的完整生命周期，自第一次调用 onCreate(Bundle)开始，直至调用 onDestroy()为止。Activity 在 onCreate()中设置所有“全局”状态以完成初始化，而在 onDestroy()中释放所有系统资源。
- Activity 的可视生命周期，自 onStart()调用开始，直到相应的 onStop()调用为止。在此期间，用户可以在屏幕上看到此 Activity，尽管它也许并没有位于前台或者正在与用户做交互。

- Activity 的前台生命周期，自 onResume()调用起，直至相应的 onPause()调用为止。在此期间，Activity 位于前台最上面并与用户进行交互。

2.2.2 Activity 生命周期实例

接下来我们就结合实例详解 Activity 生命周期的每一个过程，让读者能够更加深刻地体会 Activity 的生命周期。创建一个 Activity 并命名为 LifeActivity，如下所示：

```
package com.buaa.activitylife;

import android.app.Activity;
import android.os.Bundle;
import android.util.Log;

public class LifeActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_life);
        Log.i("LifeActivity", "onCreate");
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
        Log.i("LifeActivity", "onDestroy");
    }

    @Override
    protected void onPause() {
        super.onPause();
        Log.i("LifeActivity", "onPause");
    }

    @Override
    protected void onRestart() {
        super.onRestart();
        Log.i("LifeActivity", "onRestart");
    }

    @Override
    protected void onResume() {
        super.onResume();
```

```
            Log.i("LifeActivity", "onResume");
        }

        @Override
        protected void onStart() {
            super.onStart();
            Log.i("LifeActivity", "onStart");
        }

        @Override
        protected void onStop() {
            super.onStop();
            Log.i("LifeActivity", "onStop");
        }

    @Override
        public void onWindowFocusChanged(boolean hasFocus) {
            super.onWindowFocusChanged(hasFocus);
            Log.i("LifeActivity", "onWindowFocusChanged");
        }

        @Override
        protected void onSaveInstanceState(Bundle outState) {
            Log.i("LifeActivity", "onSaveInstanceState");
            super.onSaveInstanceState(outState);
        }

        @Override
        protected void onRestoreInstanceState(Bundle savedInstanceState) {
            Log.i("LifeActivity", "onRestoreInstanceState");
            super.onRestoreInstanceState(savedInstanceState);
        }
    }
```

在上述代码中，除了常用的 7 种事件方法外，还使用了 onWindowFocusChanged、onSaveInstanceState、onRestoreInstanceState 三种方法。下面分别对这 3 种方法进行介绍。

- onWindowFocusChanged 方法：在 Activity 窗口获得或失去焦点时被调用，例如创建时首次呈现在用户面前；当前 Activity 被其他 Activity 覆盖；当前 Activity 转到其他 Activity 或按 Home 键回到主屏，自身退居后台；用户退出当前 Activity。当 Activity 被创建时 onWindowFocusChanged 方法在 onResume 之后被调用，当 Activity 被覆盖或者退居后台或者当前 Activity 退出时在 onPause 之后被调用。

- onSaveInstanceState：在 Activity 被覆盖或退居后台之后，系统资源不足将其杀死，此方法会被调用；在用户改变屏幕方向时，此方法会被调用；在当前 Activity 跳转到其他 Activity 或者按 Home 键回到主屏，自身退居后台时，此方法会被调用。第一种情况不知道会在什么时候发生，系统根据资源紧张程度去调度；第二种情况出现在屏幕翻转方向时，系统先销毁当前的 Activity，然后重建一个新的，调用此方法时，我们可以保存一些临时数据；第三种情况下调用此方法是为了保存当前窗口各个 View 组件的状态。onSaveInstanceState 调用在 onPause 之前。
- onRestoreInstanceState：在 Activity 被覆盖或退居后台之后，系统资源不足将其杀死，然后用户又回到 Activity 时，此方法会被调用；在用户改变屏幕方向时，重建的过程中，此方法会被调用。我们可以重写此方法，以便恢复一些临时数据。onRestoreInstanceState 调用在 onStart 之后。

这 3 种方法在 Activity 的生命周期中也发挥着重要的作用，在测试时读者可以更加清晰地看出来。

此外，还需要向读者说明上述代码中的 android.util.Log 类的使用。Log 类是 Android 中用来打印日志的类，常用的方法有 5 个，即 Log.v()、Log.d()、Log.i()、Log.w()及 Log.e()，根据首字母对应 VERBOSE、DEBUG、INFO、WARN、ERROR。在 Android Studio 的控制台中打开 Android monitor 界面，选择具体的 Log level，输入关键字即可查看日志。本例中使用 Log.i()来记录 Activity 的执行过程，选择 level 为 info 级别，关键字为 LifeActivity，如图 2-3 所示。



图 2-3　查看关键字为 LifeActivity 的 Log

下面我们通过运行上面的代码来直观地观察 Activity 的生命周期。

（1）运行 Activity，Log 如下：

```
30651-30651/com.buaa.activitylife I/LifeActivity: onCreate
30651-30651/com.buaa.activitylife I/LifeActivity: onStart
30651-30651/com.buaa.activitylife I/LifeActivity: onResume
30651-30651/com.buaa.activitylife I/LifeActivity: onWindowFocusChanged
```

安装运行 Activity，不做其他任何操作，系统会在调用了 onCreate 和 onStart 之后调用 onResume，这样 Activity 就进入了运行状态。在 onResume 之后，系统还调用了 onWindowFocusChanged 方法。这个方法在某种场合下非常有用，例如程序启动时想要获取特定视图组件的尺寸大小。在 onCreate 方法执行时，因为窗口 Window 对象还没创建完成，无法取得视图组件的尺寸大小，这时就需要在 onWindowFocusChanged 里获取。

（2）跳转到其他 Activity 或按 Home 键回到主屏，Log 如下：

```
30651-30651/com.buaa.activitylife I/LifeActivity: onWindowFocusChanged
30651-30651/com.buaa.activitylife I/LifeActivity: onPause
30651-30651/com.buaa.activitylife I/LifeActivity: onSaveInstanceState
30651-30651/com.buaa.activitylife I/LifeActivity: onStop
```

退居后台时，Activity 的窗口焦点发生变化，onWindowFocusChanged 首先执行，然后才调用 onPause，之后调用 onSaveInstanceState 方法，最后调用 onStop 方法。按 Home 键回到主屏测试，跳转到其他 Activity 的结果是一样的。等学习完多个 Activity 之间跳转之后，读者可以自行测试。

（3）从后台进入前台，Log 如下：

```
30651-30651/com.buaa.activitylife I/LifeActivity: onRestart
30651-30651/com.buaa.activitylife I/LifeActivity: onStart
30651-30651/com.buaa.activitylife I/LifeActivity: onResume
30651-30651/com.buaa.activitylife I/LifeActivity: onWindowFocusChanged
```

当从后台回到前台时，系统先调用 onRestart 方法，然后调用 onStart 方法，接着调用 onResume 方法，Activity 又进入运行状态，Activity 获得焦点，调用 onWindowFocusChanged 方法。

（4）退出程序，Log 如下：

```
30651-30651/com.buaa.activitylife I/LifeActivity: onWindowFocusChanged
30651-30651/com.buaa.activitylife I/LifeActivity: onPause
30651-30651/com.buaa.activitylife I/LifeActivity: onStop
30651-30651/com.buaa.activitylife I/LifeActivity: onDestroy
```

退出程序调用了 onDestroy 方法。

（5）横屏，Log 如下：

```
30651-30651/com.buaa.activitylife I/LifeActivity: onPause
30651-30651/com.buaa.activitylife I/LifeActivity: onSaveInstanceState
30651-30651/com.buaa.activitylife I/LifeActivity: onStop
30651-30651/com.buaa.activitylife I/LifeActivity: onDestroy
30651-30651/com.buaa.activitylife I/LifeActivity: onCreate
30651-30651/com.buaa.activitylife I/LifeActivity: onStart
30651-30651/com.buaa.activitylife I/LifeActivity: onRestoreInstanceState
30651-30651/com.buaa.activitylife I/LifeActivity: onResume
30651-30651/com.buaa.activitylife I/LifeActivity: onWindowFocusChanged
```

通过日志可以发现，一次横屏的过程其实经历了一次调用 onDestroy 方法销毁 Activity 以及一次调用 onStart 方法打开 Activity 的过程，只是在这个过程中多了一次调用 onSaveInstanceState 方法保存临时数据，以及调用 onRestoreInstanceState 方法恢复数据的过程。这个发现非常有用，例如在一个视频 App 中，当用户正好观看到了一半视频时，切换了屏幕，如果不在 onSaveInstanceState 方法中保存用户观看的数据信息并在 onRestoreInstanceState 方法中恢复就会造成视频无法准确定位到用户退出时看到的位置这一尴尬状况。

当然，有时为了省去这些麻烦，开发者也可以指定 Activity 的屏幕方向，这样就不会存在屏幕切换的生命周期问题了。我们可以为 Activity 指定一个特定的方向，指定之后即使转动屏幕方向，显示方向也不会跟着改变。在 AndroidManifest.xml 中对指定的 Activity 设置 android:screenOrientation="portrait"，其中 portrait 为竖屏，landscape 为横屏，或者在 onCreate 使用 setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_PORTRAIT)方法指定，指定 ActivityInfo.SCREEN_ORIENTATION_PORTRAIT 为竖屏、ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE 为横屏。

2.3 Intent 与 Activity 之间的跳转

一个 Android 应用不可能只有一个页面，也就意味着不可能只有一个 Activity。既然存在多个 Activity，那么多个 Activity 之间是如何通信的呢？在 Android 中提供了 Intent 机制来完成组件之间的通信，本节只讲述如何使用 Intent 完成 Activity 之间的通信，其他组件之间的通信等讲述到具体内容时再做叙述。

2.3.1 Intent 简介

Intent 的中文意思是“意图，意向”，在 Android 中提供了 Intent 机制来协助应用间的交互与通信，Intent 负责对应用中一次操作的动作、动作涉及数据、附加数据进行描述，Android 则负责根据此 Intent 的描述找到对应的组件，将 Intent 传递给调用的组件，并完成组件的调用。Intent 不仅可用于应用程序之间，也可用于应用程序内部的 Activity/Service 之间的交互。因此，可以将 Intent 理解为不同组件之间通信的“媒介”，专门提供组件互相调用的相关信息。

归纳起来，Intent 的应用场合主要有以下 3 种：

- 启动一个 Activity。第一种方法是 Activity 调用 startActivity(Intent intent)直接开启一个 Activity，第二种方法是通过 Activity 调用 startActivityForResult(Intent intent, int requestCode)启动一个带请求码的 Activity，当该 Activity 结束时将回调原 Activity 的 onActivityResult()方法，并返回一个结果码。
- 启动一个 Service，等讲述到具体内容时再做叙述。
- 启动一个 Broadcast，等讲述到具体内容时再做叙述。

当使用一个 Intent 进行组件通信时，需要先实例化一个 Intent 对象，这时需要设置 Intent 的属性。Intent 的属性设置包括 Action（要执行的动作）、Data（执行动作所操作的数据）、Type（显式指定 Intent 的数据类型）、Category（执行动作的附加信息）、Component（指定 Intent 目标组件的类名称）、Extras（其他所有附加信息的集合）。下面具体叙述这些属性。

- Action：在 SDK 中定义了一系列标准动作，其中的一部分如图 2-4 所示。

onstant	Target component	Action
ACTION_CALL	activity	Initiate a phone call.
ACTION_EDIT	activity	Display data for the user to edit.
ACTION_MAIN	activity	Start up as the initial activity of a task, with no data input and no returned output.
ACTION_SYNC	activity	Synchronize data on a server with data on the mobile device.
ACTION_BATTERY_LOW	broadcast receiver	A warning that the battery is low.
ACTION_HEADSET_PLUG	broadcast receiver	A headset has been plugged into the device, or unplugged from it.
ACTION_SCREEN_ON	broadcast receiver	The screen has been turned on.
ACTION_TIMEZONE_CHANGED	broadcast receiver	The setting for the time zone has changed.

图 2-4 SDK 的部分标准动作

其中，ACTION_CALL 表示调用拨打电话的应用，ACTION_EDIT 表示调用编辑器，ACTION_SYNC 表示同步数据。

- Data：在 Intent 中，Data 使用指向数据的 URI 来表示。比如，在联系人应用中，指向联系人列表的 URI 是 content://contacts/people/。
- Type：对于不同的动作，其 URI 数据的类型是不同的。通常，Intent 的数据类型能够根据数据本身进行判定，但是通过设置这个属性可以强制采用显式指定的类型。
- Category：Category 表示执行动作的附加信息。比如，当我们想要让所执行的动作被接收后，作为顶级应用而位于其他所有应用的最上层，并可以使用附加信息 LAUNCHER_CATEGORY 来实现。
- Component：用于指定 Intent 目标组件的类名称。通常，Android 会根据 Intent 中所包含的其他属性信息（比如 Action、Data/Type、Category）进行查找，并找到一个与之匹配的目标组件。但是，如果我们设置了 Component 属性，明确指定了 Intent 目标组件的类名称，那么上述查找过程将不需要执行。
- Extras：可以为组件提供扩展信息。

另外，在使用 Intent 时，根据是否明确指定 Intent 对象的接收者，可以分为两种情况。一种是显式的 Intent，即在构造 Intent 对象时就指定接收者；另一种是隐式的 Intent，即在构造 Intent 对象时并不指定接收者。

对于显式的 Intent 来说，Android 不需要解析 Intent，因为目标组件已经很明确。对于隐式的 Intent 来说，Android 需要对其进行解析，通过解析将 Intent 映射给可以处理该 Intent 的 Activity、Service 或 Broadcast。

Intent 解析机制是通过查找注册在 AndroidManifest.xml 文件中的所有 IntentFilter 以及 IntentFilter 所定义的 Intent 找到最匹配的 Intent。

在解析过程中，Android 通过判断 Intent 的 Action、Type、Category 这 3 个属性找出最匹配的 Intent，具体的判断方法如下：

（1）如果 Intent 指明了 Action，那么目标组件 IntentFilter 的 Action 列表中就必须包含有这个 Action，否则不能匹配。

（2）如果 Intent 没有提供 Type，那么系统将从 Data 中得到数据类型。目标组件的数据类型列表中必须包含 Intent 的数据类型，否则不能匹配。

（3）如果 Intent 中的数据不是 content: URI，而且 Intent 也没有明确指定它的 Type，就将根据 Intent 中数据的 scheme（比如 http: 或者 mailto:）进行匹配。同理，Intent 的 scheme 必须出现在目标组件的 scheme 列表中，否则不能匹配。

（4）如果 Intent 指定了一个或多个 Category，那么这些类别必须全部出现在目标组件的类别列表中，否则不能匹配。

2.3.2 使用 Intent 进行 Activity 跳转

下面我们通过新建一个包含两个 Activity 的 Android 工程来实现应用程序内部之间 Activity 的跳转。除了系统生成的 MainActivity，我们再手动新建一个 SecondaryActivity。建立

Activity 的方法是到需要的包下右击，然后单击 new→Activity，选择需要的 Activity 类型即可。此时 Android Studio 会自动在 AndroidManifest.xml 中注册，无须手动注册。

1. 显式意图

显式意图要求必须知道被激活组件的包和 class。如下代码便实现了从 MainActivity 跳转到 SecondaryActivity，并向 SecondaryActivity 中传递一个字符串的功能。

```
package com.buaa.intent;

import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;

public class MainActivity extends AppCompatActivity {

    private Button button;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        button = (Button)findViewById(R.id.button);
        button.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                //跳转到第二个 Activity
                gotoSecondaryActivity();
            }
        });
    }
    private void gotoSecondaryActivity(){
        Intent toSecondary = new Intent();                          //创建一个意图
        toSecondary.setClass(this, SecondaryActivity.class);        //指定跳转 SecondaryActivity
        toSecondary.putExtra("string", "Ricky");                    //设置传递字符串
        toSecondary.putExtra("int", 25);                            //设置传递 int 类型内容
        startActivity(toSecondary);
    }
}
```

对 activity_main.xml 做一些改造：

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
```

```
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:paddingBottom="@dimen/activity_vertical_margin"
        android:paddingLeft="@dimen/activity_horizontal_margin"
        android:paddingRight="@dimen/activity_horizontal_margin"
        android:paddingTop="@dimen/activity_vertical_margin"
        tools:context="com.buaa.intent.MainActivity">

    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAllCaps="false"
        android:text="跳转到第二个 Activity" />
    </RelativeLayout>
```

这里为了跳转方便，使用了一个 button 按钮，其中 findViewById 方法已经在前文讲过，是获得 View 控件对象的方法。button.setOnClickListener()是事件处理的内容，这里不做详细解释，读者只需要像文中一样使用即可，之后重点讲解如何进行事件处理的内容，现在重点关注 gotoSecondaryActivity 方法。gotoSecondaryActivity 方法中的代码使用的各种方法在上一部分有讲述，而注释部分也说明了它们的作用，此处不再分析。仔细分析会发现，使用 Intent 进行 Activity 跳转分为两步，第一步构建 Intent 的对象，第二步调用 startActivity 方法开启第二个 Activity。除了上面给出的构建 Intent 的对象方法外，还可以直接在实例化 Intent 对象时在 Intent 的构造函数中传入第二个 Activity 类，或者使用 setComponent 方法传入。而向第二个 Activity 传递数据除了上面直接用 intent 对象调用 putExtra 方法外，还可以使用如下方法：

```
Bundle bundle = new Bundle();
bundle.putString("name", "Ricky");
bundle.putInt("age", 25);
toSecondary.putExtras(bundle);
```

上面给出了如何实现跳转的方法，那么如何在 SecondaryActivity 中接收数据呢？代码如下：

```
package com.buaa.intent;

import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.util.Log;

public class SecondaryActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
```

```
            setContentView(R.layout.activity_secondary);

            Intent intent_accept = getIntent();             //创建一个接收意图
            Bundle bundle = intent_accept.getExtras();      //创建 Bundle 对象，用于接收 Intent 数据
            String name = bundle.getString("name");         //获取 Intent 的内容 name
            int age = bundle.getInt("age");                 //获取 Intent 的内容 age
            Log.i("SecondaryActivity",name+"    "+age);
        }
}
```

为了方便辨识，改造 activity_secondary.xml 文件：

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="com.buaa.intent.SecondaryActivity">

    <TextView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:text="第二个 Activity"/>
</RelativeLayout>
```

接收前一个 Activity 传递来的参数也需要使用 Intent。具体来说就是创建 Intent 对象，使用 intent 对象来获得一个 bundle 实例，传递的参数就存储在 bundle 实例中。这里为了方便使用 Log 来打印接收过来的数据。

运行程序，如图 2-5 所示。

点击按钮就会跳转到第二个 Activity，如图 2-6 所示。



图 2-5　程序运行显示效果

图 2-6　第二个 Activity 显示的界面

查看 Log 会发现有如下记录：

```
497-2497/com.buaa.intent I/SecondaryActivity: Ricky  25
```

这样就说明应用成功地执行了两个 Activity 之间的跳转，并且在两个 Activity 之间进行了数据的传输。

2. 隐式意图

隐式意图和显式意图不同，它可以不知道被激活组件的包和 class，只通过指定 action 就进行跳转。同时，被激活的组件必须是在 AndroidManifest.xml 文件中注册的，注册的方式如下：

```
<activity android:name=".SecondaryActivity">
    <intent-filter>
        <action android:name="com.buaa.SecondaryActivity"/>
        <category android:name="android.intent.category.DEFAULT"/>
    </intent-filter>
</activity>
```

其中的重点就是加入：

```
<intent-filter>
        <action android:name="com.buaa.SecondaryActivity"/>//这里可以根据需要设置 name
        <category android:name="android.intent.category.DEFAULT"/>
    </intent-filter>
```

除此之外，和显式意图相比，只需要将 gotoSecondaryActivity 方法中的 toSecondary.setClass (this, SecondaryActivity.class)改为 toSecondary.setAction("com.buaa.SecondaryActivity")即可，这里的参数内容需要和 AndroidManifest.xml 文件中注册的内容一致，其他部分都不需要改动。

```
72/com.buaa.intent I/SecondaryActivity: Ricky  25
```

测试的结果也是一样的。读者可能会问，它们如此相像，为什么还需要两种呢？简单来说就是隐式意图可以更好地让代码解耦，使不同模块之间的耦合性更低。另外，如果一个 Activity 想要启动另一个应用中的 Activity 就只能使用隐式意图。

3. 带回调方法的意图

有时我们需要通过定义在 MainActivity 中的某一控件来启动 SecondaryActivity，并且当 SecondaryActivity 结束时返给 MainActivity 一个执行结果。要实现上述功能，只需要完成以下 3 步即可。

步骤 01 在 MainActivity 中实现向 SecondaryActivity 发送带请求码的意图，具体实现方法如下（改造 MainActivity 中的 gotoSecondaryActivity 方法即可）：

```
private int RequestCode=1023;//设置请求码
private void gotoSecondaryActivity(){
    Intent toSecondary = new Intent();                         //创建一个意图
    toSecondary.setClass(this, SecondaryActivity.class);       //指定跳转到 SecondaryActivity
    startActivityForResult(toSecondary, RequestCode);          //启动带请求码意图
}
```

步骤 02 在 SecondaryActivity 中接收 toSecondary _request，并向意图中填充要返给 MainActivity 的内容，最后还需要设置一个返回码。加入一个 button 按钮，并实现点击时结束 SecondaryActivity。改造 SecondaryActivity：

```
public class SecondaryActivity extends AppCompatActivity {
    private Button finishButton;
    public   static final int RESULT_CODE = 2003;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_secondary);

        finishButton = (Button) findViewById(R.id.finish);
        finishButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Intent intent = getIntent();                 //创建一个接收意图
                intent.putExtra("name", "ricky");            //设置意图的内容
                setResult(RESULT_CODE, intent);              //设置结果码
                finish();                                    //结束 SecondaryActivity，返回 MainActivity
            }
        });
    }
}
```

在 activity_secondary.xml 文件中加入一个 button 按钮：

```
<Button
    android:id="@+id/finish"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="结束二个 Activity" />
```

步骤 03 结束 SecondaryActivity 时将返回到 MainActivity 界面。此时，MainActivity 中的 onActivityResult()方法将被回调。在本示例中，该方法的具体实现如下：

```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    if(requestCode == RequestCode && resultCode == SecondaryActivity.RESULT_CODE) {
        Bundle bundle = data.getExtras();
        String name = bundle.getString("name");
        Log.i("MainActivity",name);
    }
}
```

经过上述的 3 个步骤，运行应用，第二个 Activity 中会出现一个按钮，点击这个按钮就回到了第一个 Activity 中。此时在控制台上打开日志，会发现有如下记录：

```
19627-19627/com.buaa.intent I/MainActivity: ricky
```

这样应用就完成了一次带回调方法的跳转。

4. 跳转中对象参数的传递

在 Android 开发中，有时多个 Activity 之间需要进行对象的传递，使用 Intent 也可以完成这一功能。具体来说就是将显式跳转或者隐式跳转中的例子做如下修改。

先创建一个 User 类：

```
package com.buaa.intent;
public class User  implements Serializable{
    private String name;
    private int age;
    public int getAge() {
        return age;
    }
    public void setAge(int age) {
        this.age = age;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
}
```

再改造 MainActivity 中的 gotoSecondaryActivity 方法：

```
private void gotoSecondaryActivity(){
    Intent toSecondary = new Intent();
    toSecondary.setAction("com.buaa.SecondaryActivity");
    Bundle bundle = new Bundle();
    User user = new User();
    user.setAge(25);
    user.setName("ricky");
    bundle.putSerializable("user",user);
    toSecondary.putExtras(bundle);
    startActivity(toSecondary);
}
```

最后改造 SecondaryActivity 中接收参数的内容为：

```
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_secondary);
        Intent intent = getIntent();
        Bundle bundle = intent.getExtras();
        User user = (User)bundle.get("user");
        Log.i("SecondaryActivity",user.getName()+"    "+user.getAge());
    }
```

通过上面的改造，一个在多个 Activity 之间传递对象参数的应用就完成了。运行应用，点击按钮即可进入第二个 Activity，并在日志中出现如下记录：

```
5223-5223/com.buaa.intent I/SecondaryActivity: ricky  25
```

和我们的预期一致，这说明通过 Intent 在多个 Activity 之间传递对象是可行的。

Activity 和 Intent 是 Android 中非常重要的内容，希望读者在学习完之后多做练习，多加揣摩。

2.4 Activity 启动模式

通过 2.3 节的学习，读者应该明白了这样一个事实，一个应用程序当中通常都会包含很多个 Activity，多个 Activity 之间还应该是可以相互启动的。在 Activity 启动时是有不同模式的，本节将重点讲解 Activity 的 4 种启动模式。

启动模式在多个 Activity 跳转的过程中扮演着重要的角色，可以决定是否生成新的 Activity 实例、是否重用已存在的 Activity 实例、是否和其他 Activity 实例共用一个 Task。Task 是与 Activity 相关的一个重要概念，密切联系着 Activity 栈，是一组以栈的模式聚集在一起的 Activity 组件集合，Task 以栈来管理 Activity。一个 Task 可以管理多个 Activity，启动一个应用，也就创建了一个与之对应的 Task。简单地说就是 Activity 是 Task 用栈的方式进行管理的。

在 Android 中 Activity 包括 standard、singleTop、singleTask 以及 singleInstance 四种启动模式。我们可以在 AndroidManifest.xml 中配置<activity>的 android:launchMode 属性为以上任一种模式。下面我们结合实例一一介绍这 4 种启动模式。

2.4.1 standard 模式

standard 是默认的启动模式，如果不指定 launchMode 属性，就会自动使用这种启动模式。创建一个 Android 工程，新建一个 MainActivity 类来具体分析，代码如下：

```
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
```

```
            setContentView(R.layout.activity_main);

            TextView textView = (TextView) findViewById(R.id.text);
            textView.setText(this.toString());
            Button button = (Button) findViewById(R.id.button);
            button.setOnClickListener(new View.OnClickListener() {
                @Override
                public void onClick(View v) {
                    Intent intent = new Intent(MainActivity.this, MainActivity.class);
                    startActivity(intent);
                }
            });
        }
    }
```

布局文件代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context="com.buaa.launchmode.MainActivity">
    <TextView
        android:id="@+id/text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!" />
    <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="进入下一个 Activity"
        android:id="@+id/button"/>
</LinearLayout>
```

本例中使用了 button 按钮和 textView 文本框，读者知道即可，下面章节会有具体讲解。这里主要看代码中的关键部分，在 MainActivity 中使用 Intent 显式启动了 MainActivity，并显示当前 Activity 对象的 hash 值。运行程序，进入的 Activity 界面如图 2-7 所示。

点击 MAINACTIVITY 按钮，进入的界面如图 2-8 所示。

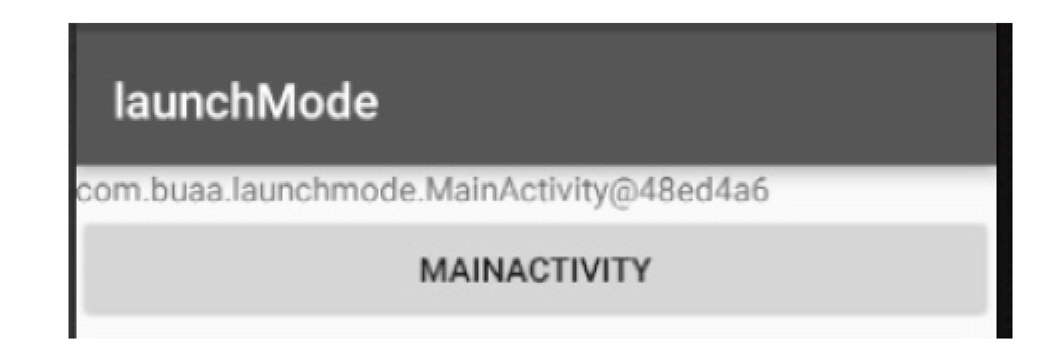


图 2-7　原始的 MainActivity 界面

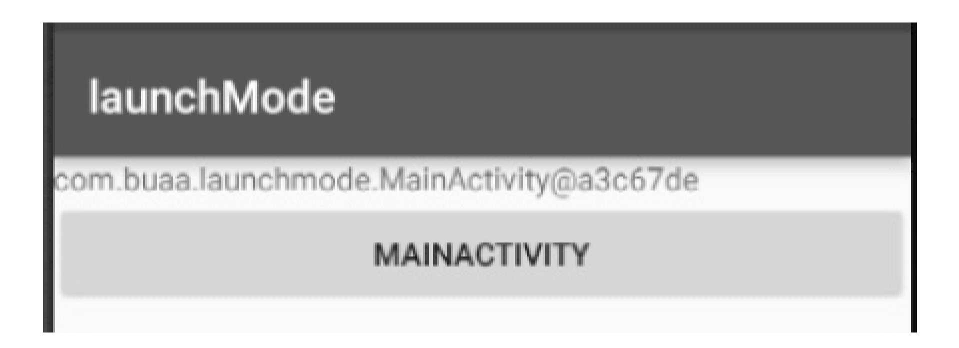


图 2-8　跳转之后的 MainActivity 界面

接着点击 MAINACTIVITY 按钮，进入的界面如图 2-9 所示。



图 2-9　继续跳转之后的 MainActivity 界面

不管点击多少次，虽然显示的都是 MainActivity 对象，但是都是不同对象。这种启动模式表示每次启动该 Activity 时都会创建一个新的实例，并且总会把它放入当前的任务当中。声明为这种启动模式的 Activity 可以被实例化多次，一个任务当中也可以包含多个 Activity 的实例。

2.4.2　singleTop 模式

直接使用上面的例子，并在 AndroidManifest.xml 中配置<activity>的 android:launchMode 属性为 singleTop，代码如下：

```
<activity android:name=".MainActivity"
    android:launchMode="singleTop">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />

        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

运行程序，会进入如图 2-10 所示的界面。

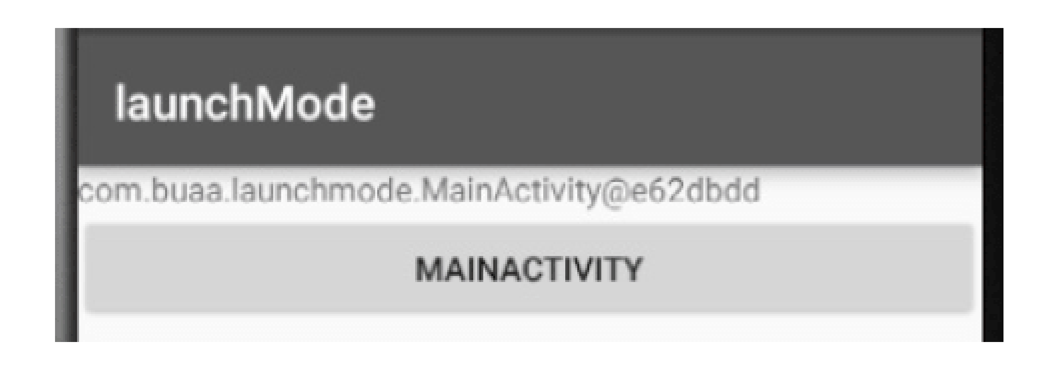


图 2-10　当 MainActivity 处于栈顶时界面的状态

此时，不管点击多少次按钮，打印的 MainActivity 对象都是同一个。此时，有的读者可能会说使用 singleTop 模式启动的 Activity 不会创建多个实例，使用的都是同一个 Activity 实例。其实这是不对的。下面我们再来看另一个例子。在这个例子中，我们再创建一个 OtherActivity，代码如下：

```
public class OtherActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_other);
```

```
            TextView textView = (TextView) findViewById(R.id.otherText);
            textView.setText(this.toString());
            Button button = (Button) findViewById(R.id.otherButton);
            button.setOnClickListener(new View.OnClickListener() {
                @Override
                public void onClick(View v) {
                    Intent intent = new Intent(OtherActivity.this, MainActivity.class);
                    startActivity(intent);
                }
            });
        }
    }
```

布局文件和 MainActivity 的布局文件基本一致，只需要将 textView 和 Button 对应的 Id 修改为和上述代码中的 Id 一致即可。

另外，需要将 MainActivity 中的 Intent 部分代码修改为：

```
Intent intent = new Intent(MainActivity.this,
OtherActivity.class);
```

运行程序，点击按钮从 MainActivity 先跳转到 OtherActivity，再点击按钮跳转到 MainActivity，过程如图 2-11 所示。

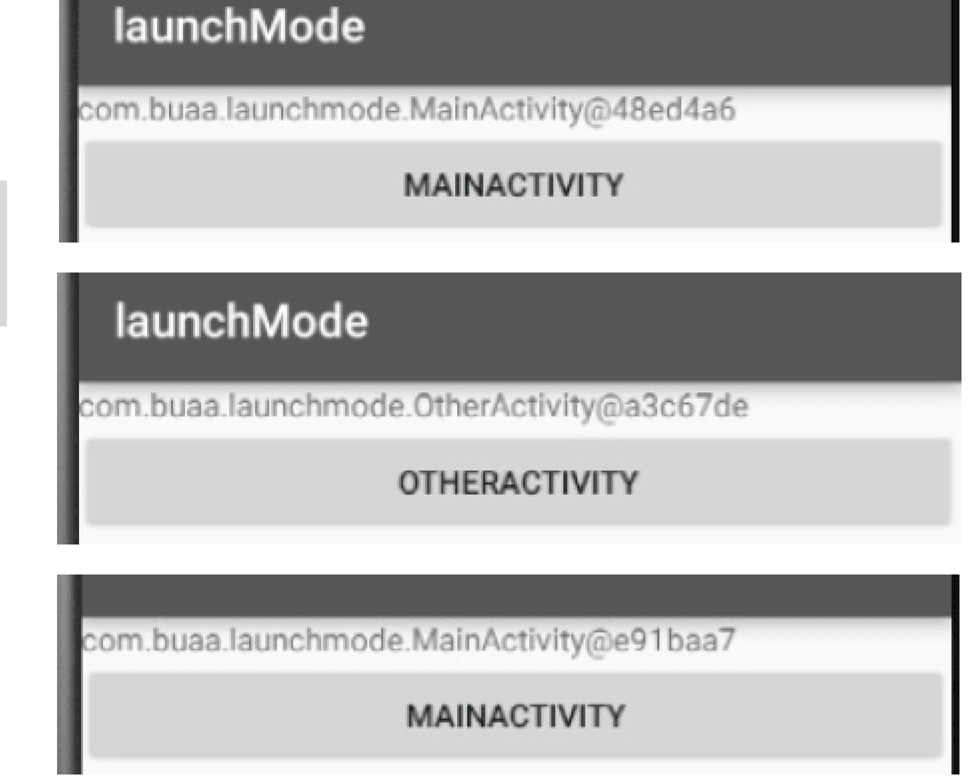


图 2-11　MainActivity 不处于栈顶时的状态

从图 2-11 中可以清晰地发现，MainActivity 创建了不同的实例。那么为什么同样是使用 singleTop 模式，在上一个例子中 MainActivity 只创建了一个实例呢？singleTop 模式的启动方式到底是什么样的呢？

其实，对于 singleTop 模式，如果要启动的这个 Activity 在当前任务中已经存在了，并且还处于栈顶的位置，那么系统就不会再去创建一个该 Activity 的实例，而是调用栈顶 Activity 的 onNewIntent()方法（读者在练习时可以进行观察，这里不做分析了）。声明成这种启动模式的 Activity 也可以被实例化多次，一个任务当中也可以包含多个这种 Activity 的实例。

举个例子来讲，一个 Task 的返回栈中有 A、B、C、D 四个 Activity，其中 A 在最底端，D 在最顶端。这时如果我们要求再启动一次 D，并且 D 的启动模式是 standard，那么系统就会再创建一个 D 的实例放入返回栈中，此时栈内元素为：A-B-C-D-D。而如果 D 的启动模式是 singleTop，由于 D 已经是在栈顶了，那么系统就不会再创建一个 D 的实例，而是直接调用 D Activity 的 onNewIntent()方法，此时栈内元素仍然为：A-B-C-D。

2.4.3　singleTask 模式

在上面例子的基础上，将 AndroidManifest.xml 文件中 MainActivity 与 OtherActivity 的 launchMode 属性修改为 android:launchMode="singleTask"。运行程序，多次点击按钮，过程如图 2-12 所示。

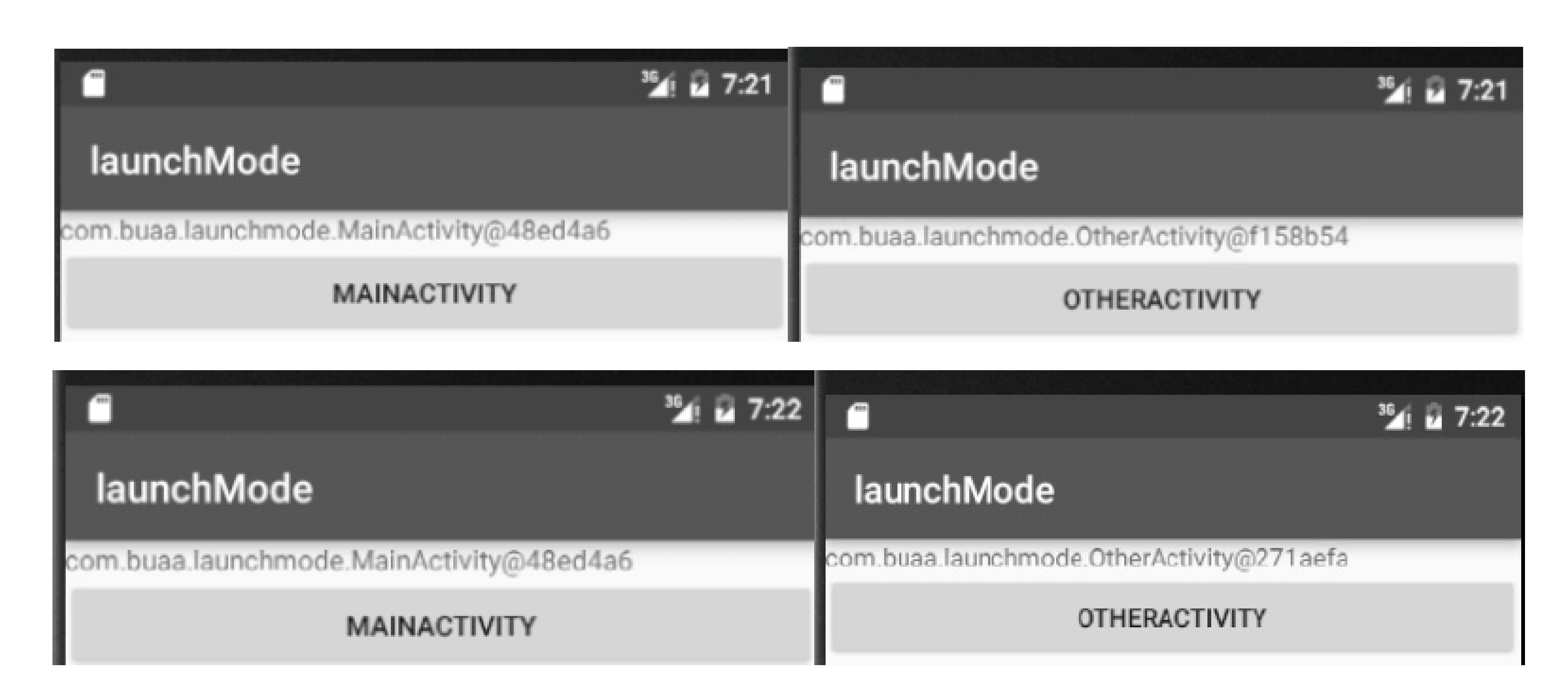


图 2-12 singleTask 模式下 Activity 启动状态

显然，声明了使用 singleTask 模式启动的 MainActivity 只创建了一个实例，虽然 MainActivity 并没有位于 Task 返回栈的栈顶，但是同样使用了 singleTask 模式启动的 OtherActivity 却创建了新的实例。

singleTask 这种启动模式表示，系统会创建一个新的任务，并将启动的 Activity 放入这个新任务的栈底位置。但是，如果现有任务当中已经存在一个该 Activity 的实例了，那么系统就不会再创建一次它的实例，而是会直接调用 onNewIntent()方法。声明成这种启动模式的 Activity 在同一个任务当中只会存在一个实例。

读者会很疑惑，因为按照上述解释，OtherActivity 也不应该创建新的实例。这是因为这里我们所说的启动 Activity 都指的是启动其他应用程序中的 Activity，因为 singleTask 模式在默认情况下只有启动其他程序的 Activity 才会创建一个新的任务，启动自己程序中的 Activity 还是会使用相同的任务。如果启动的对象是本应用内的 Activity，那么如果发现有对应的 Activity 实例，就使此 Activity 实例之上的其他 Activity 实例统统出栈，使此 Activity 实例成为栈顶对象，显示到幕前。本例中，OtherActivity 启动 MainActivity 时，发现在 Task 的返回栈中有对应的 MainActivity 实例，于是把它上面的其他 Activity 实例都推出栈，它成为栈顶对象，OtherActivity 也因此被推出了栈。所以当第二次启动 OtherActivity 时，并没有在 Task 的返回栈中找到对应的实例，只能创建一个新的实例。很多 Android 书籍中并未指出此种情况，读者需要多加注意。

2.4.4 singleInstance 模式

使用 singleInstance 模式启动 Activity 会先创建一个新的 Task，这种 Activity 所在的 Task 中始终只会有一个 Activity，通过这个 Activity 打开的其他 Activity 会被放入到别的任务当中。

修改 MainActivity 的 launchMode 属性为 android:launchMode="standard"，修改 OtherActivity 的 launchMode 属性为 android:launchMode="singleInstance"。为了能够展示出使用 singleInstance 模式启动 Activity 会先创建一个新的 Task 这种效果，加入如下代码以展示 taskId：

```
TextView otherText = (TextView) findViewById(R.id.taskIdMain);
otherText.setText("当前 task 的 ID 为："+this.getTaskId())
```

在布局文件中加入如下代码：

```
<TextView
```

```
        android:id="@+id/taskIdMain"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
```

这是在 MainActivity 中的实例，在 OtherActivity 中修改的内容除了 Id 以外，和 MainActivity 中的一样。

运行程序，从 MainActivity 中跳转到 OtherActivity，再到 MainActivity，再到 OtherActivity，效果如图 2-13 所示。

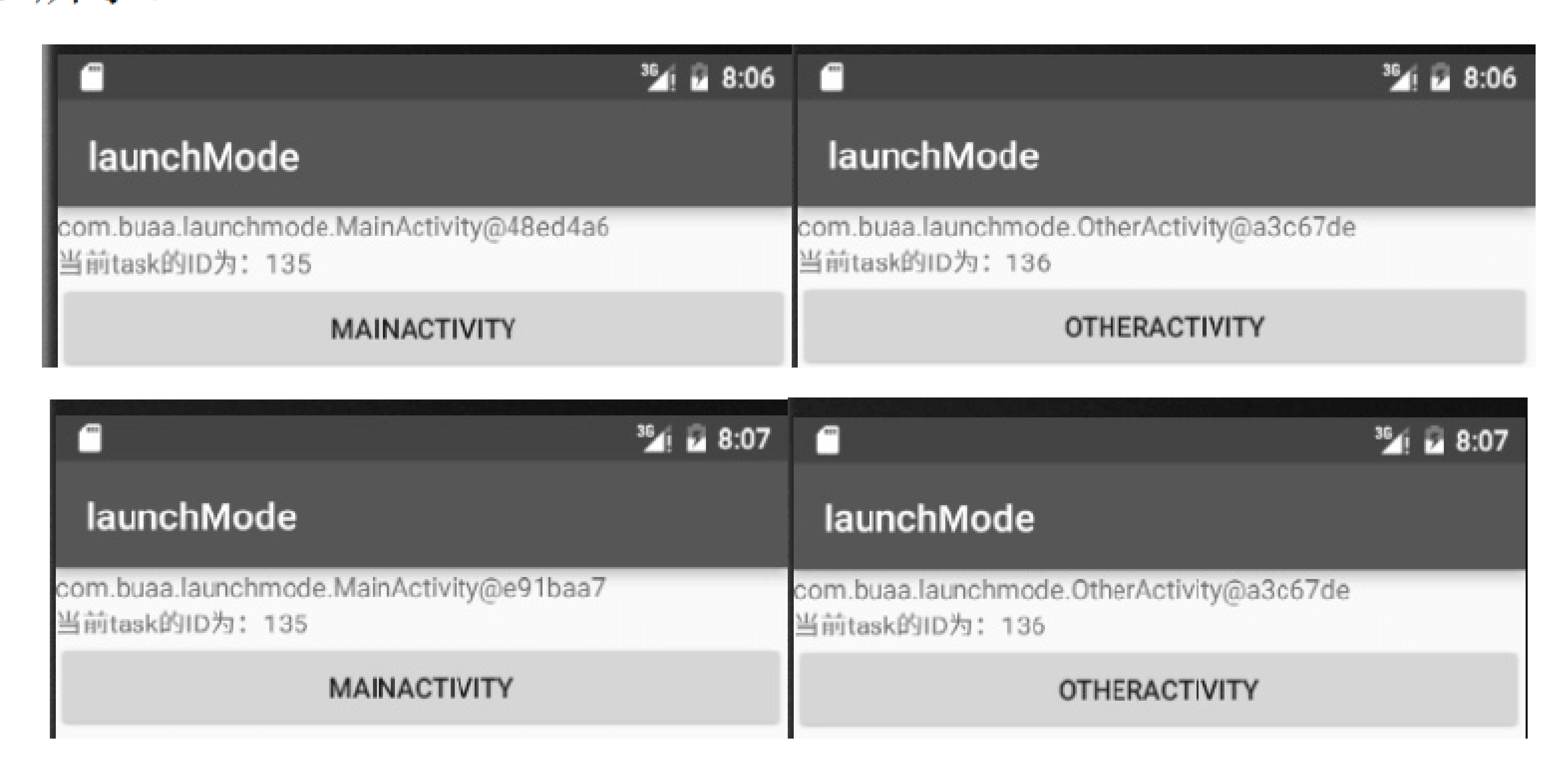


图 2-13　singleInstance 模式下 Activity 启动状态

可以很清晰地发现，系统确实为使用了 singleInstance 模式启动的 OtherActivity 创建了新的 Task。而且在这个 Task 返回栈中的 Activity 实例是同一个，当它在去启动 MainActivity 时，MainActivity 依旧进入了 ID 为 135 的 Task 中，并没有留在当前的 Task 中。

按 Back 键第一次返回时，会进入上一个界面，如图 2-14 所示。

再按一次返回键后退时并不会返回 OtherActivity，而是直接返回到上一个 MainActivity，如图 2-15 所示。

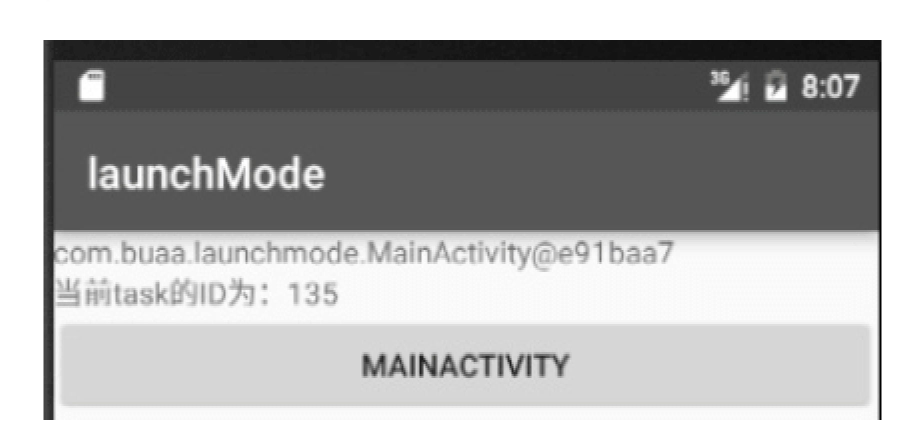



图 2-14　第一次按返回键时的界面

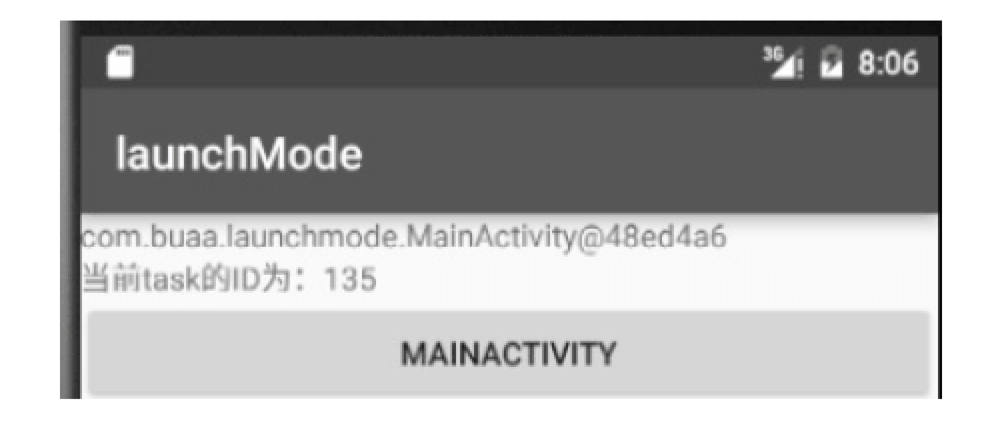


图 2-15　第二次按返回键时的界面

这里读者可能会提出为什么没有退到第二次的 OtherActivity 中呢？这是因为，在 Id 为 135 的 Task 中创建了两个 MainActivity 实例，由于使用了 singleInstance 模式，因此在 Id 为 136 的 Task 中只有一个 OtherActivity 实例，所以当在 OtherActivity 界面按返回键后当前 Task 的返回栈空了，应用回到了 Id 为 135 的 Task 中，此时第二次出现的 MainActivity 实例处于栈顶并显示在界面中，再按返回键，最初的 MainActivity 实例进入栈顶显示在界面中。如果此时再按返回键，当前 Task 中的返回栈也将为空，退出应用。

使用 singleInstance 模式的好处就是当多个应用或者 Activity 启动某个 Activity（使用 singleInstance 模式启动）时，共用同一个返回栈，解决了共享 Activity 实例的问题。

通过上面的论述，读者对 4 种启动模式应该有了较深的了解。standard 模式是 Android 默认的启动模式。但是使用 standard 模式启动 Activity 可能会造成多次启动的问题，比如用户手误多次点击一个跳转到新的 Activity 的按钮，这时系统就会创建多个新的 Activity，但是用户其实只需要一个。我们借助 singleTop 模式来避免这个问题。将 Activity 的启动模式指定为 singleTop，在启动 Activity 时如果发现返回栈的栈顶已经是该 Activity，就认为可以直接使用它，不会再创建新的 Activity 实例。如果该 Activity 并没有处于栈顶的位置，还是可能会创建多个 Activity 实例的。将 Activity 的启动模式指定为 singleTask，每次启动该 Activity 时系统首先会在返回栈中检查是否存在该 Activity 的实例，如果已经存在就直接使用该实例，并把在这个 Activity 之上的所有活动统统出栈，如果没有就会创建一个新的 Activity 实例。不同于以上 3 种启动模式，指定为 singleInstance 模式的活动会启用一个新的返回栈来管理这个活动，这样做的好处就是解决了多个应用访问一个 Activity 时的共享实例问题。

2.5 小　结

本章从第一个 Android 程序开始讲起，系统地讲解了 Activity 的概念、生命周期、多个 Activity 之间的跳转，以及 Activity 的 4 种启动模式，并介绍了 Intent 在 Activity 组件中的应用，并且讲述了如何使用 Log。

本章的内容是 Android 中至关重要的一部分，希望读者能够反复练习，熟练掌握。考虑到读者是初学 Android，同时在本书中会有教授如何进行产品开发的内容，所以一些关于 Activity 的开发使用技巧和常用的类设计模式在此处并没有讲解，而是保留到产品开发时讲解。



第 3 章 用户界面 UI 的开发

对于 Android 应用开发最基本的就是用户界面（Graphics User Interface，GUI）的开发。如果一个应用没有好的界面，就很难吸引最终用户。所以用户界面的开发对于 Android 应用开发是很重要的，是首先要掌握的。从本章开始，我们将讲解 UI 相关的内容。

Android 中的 UI 组件都继承自 android.view.View 类，所有的 UI 组件都位于 android.view 包 和 android.widget 包中，主要分为 View（视图：基本控件）和 ViewGroup（容器：布局管理器）两大类。布局管理器是本章讲解的重点。

3.1 布局管理器概述

布局管理器可以根据屏幕大小管理容器内的控件，自动适配组件在手机屏幕中的位置，所以在 Android 界面开发中布局管理器的作用非常重要。从图 3-1 中可以看出，布局管理器都是以 ViewGroup 为基类派生出来的，共有 6 种。

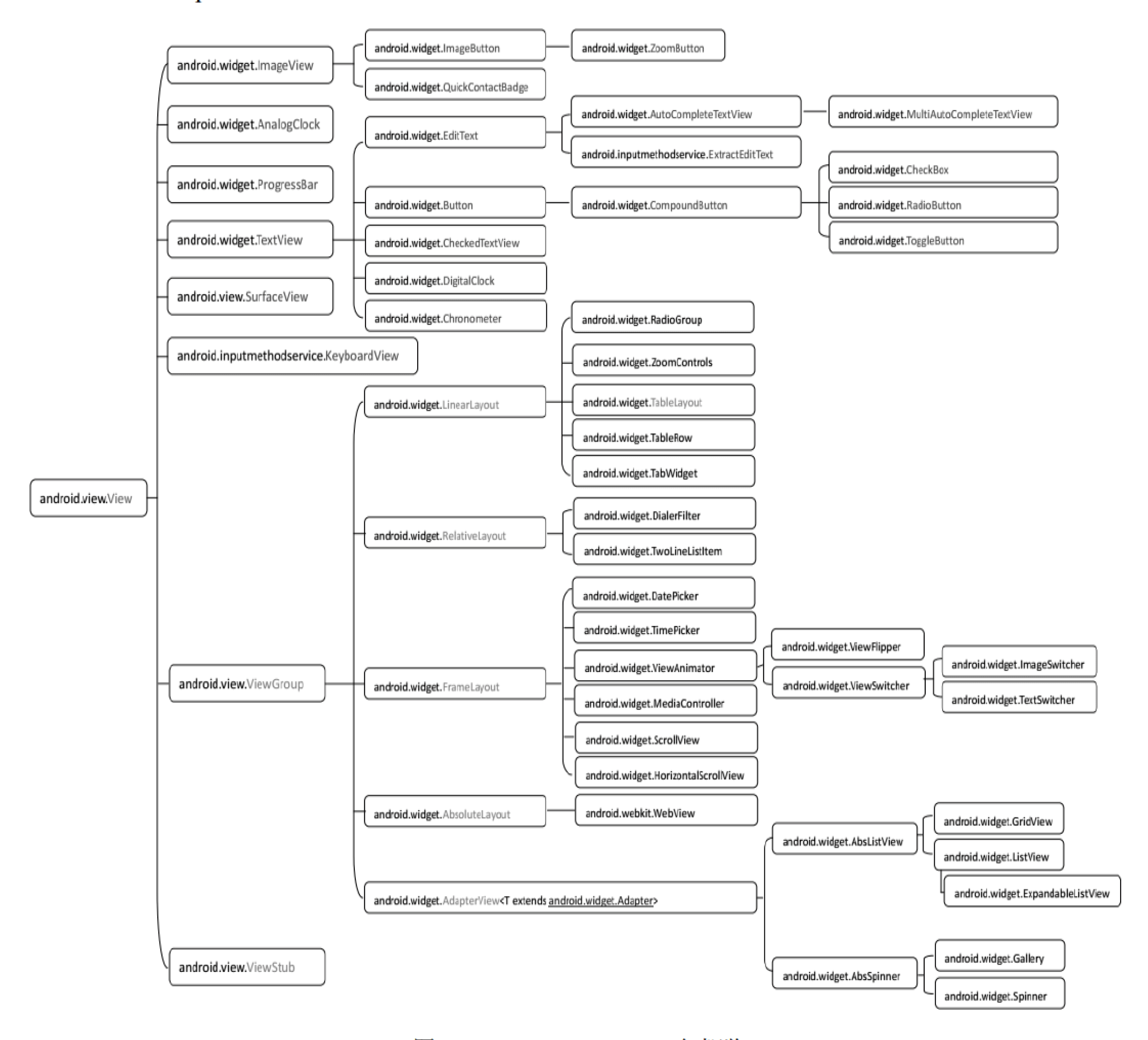


图 3-1　android.view.View 包概览

- LinearLayout：线性布局管理器，布局内的控件不换行或者换列，组件依次排列，超出容器的控件则不会被显示。
- TableLayout：表格布局管理器，继承自 LinearLayout 线性布局。表格布局管理器用行、列方式来管理容器内的控件，表格布局不需要制定多少行列，布局内每添加一行 TableRow 表示添加一行，然后在 TableRow 添加子控件，容器的列数由包含列数最多的行决定。

- RelativeLayout：相对布局管理器，是 Android studio 中默认的布局管理器。容器内的控件布局总是相对于父容器或兄弟组件的位置而定。
- FrameLayout：帧布局管理器，为容器内的控件创建一块空白区域(帧)，一帧一个控件，后面添加的控件覆盖在前面的控件上面。类似于 Java AWT 中的 CardLayout 布局。
- AbsoluteLayout：绝对布局管理器，控件的位置大小需要开发人员通过指定 X、Y 坐标来确定。
- GridLayout：网格布局管理器，是 Android 4.0 以后才增加的布局管理器，将容器划分为行×列的网格，将每个控件置于网格中，当然也可以通过设置相关属性使一个控件占据多行或多列。

从下一节开始将具体讲解如何使用这些布局管理器。在此之前，读者需要先了解一个布局文件的大致格式，以及如何去编写程序界面。下面给出一个布局文件实例：

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.buaa.moreview.MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!" />
</RelativeLayout>
```

在本节之前可能读者已经多次看过这种 xml 格式的文件。它在 layout 文件夹下，被 Activity 在 onCreate()方法中引用。其中第一行<?xml version="1.0" encoding="utf-8"?>指定 xml 的版本以及编码格式，是固定的，读者在开发时无须改动。下面是一个 RelativeLayout 标签，包裹着一个 TextView 标签，这里 RelativeLayout 标签指定这个布局文件使用的是相对布局。就目前来说，在开始时读者姑且认为布局文件就是这样一种模式，第一行指定 xml 文件的版本与编码格式，之后最外层使用 6 种布局管理器中的一种，并在布局管理器中添加各种控件，这些控件就会按照相应布局管理器的特性排列。

3.2 LinearLayout：线性布局管理器

线性布局管理器会将容器中的组件一个一个排列起来，LinearLayout 可以通过 android: orientation 属性控制组件横向或者纵向排列。线性布局中的组件不会自动换行，当组件一个一个排列到尽头之后，剩下的组件就不会显示出来了。

3.2.1 LinearLayout 实例及属性详解

LinearLayout 布局文件实例：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                   //设置布局管理器为 LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"         //设置组件宽度
    android:layout_height="match_parent"        //设置组件高度
    android:layout_gravity="center"             //设置组件位置
    android:layout_weight="1"                   //设置组件占容器的权重
    android:gravity="center"                    //设置组件中子元素的位置
    android:visibility="visible"                //设置组件的高度
    android:orientation="vertical">             //设置组件内容是横向还是竖向排列

<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="hello liruiqi" />

<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="hello liruiqi1" />

<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="hello liruiqi2" />

</LinearLayout>
```

程序中定义了 3 个文本显示控件，采用了 vertical（垂直）布局，运行效果如图 3-2 所示。

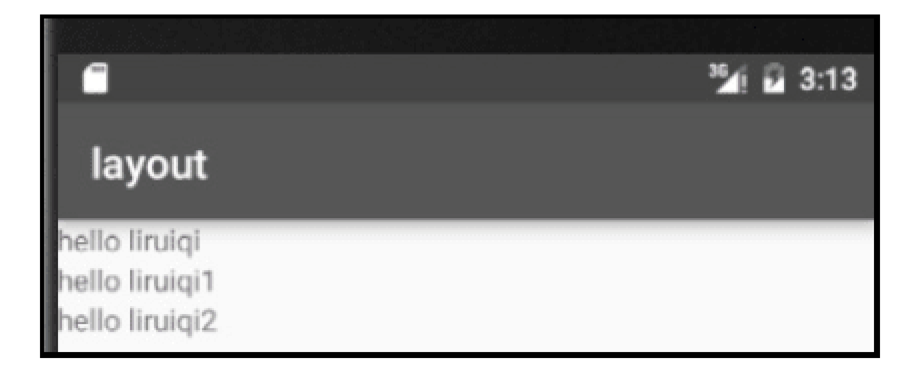


图 3-2　线性布局效果图

在上述 LinearLayout 布局文件实例中可以清晰地看出，LinearLayout 内部的子元素是按照线性布局的，也可以看出布局文件中包含了 LinearLayout 的一些常用属性：

- android:layout_width　设置当前组件的宽度，match_parent 表示充满整个父元素，若使用 wrap_content 则意味着组件多大就多大。
- android:layout_height　设置当前组件的高度，match_parent 表示充满整个父元素，若使用 wrap_content 则意味着组件多大就多大。

- android:orientation 当设置成 vertical 时表示布局容器内的控件纵向排列成 1 列，当设置成 horizontal 时表示布局容器内的所有控件横向排列成 1 行。
- android:layout_weight 为容器内的组件设置权重，表示当所有控件全部排列完毕后被设置的组件占父容器剩余空白部分的比重。
- android:layout_gravity 为容器内的控件设置该控件在父容器中的对齐方式，当父容器线性设置为 vertical 纵向时，只有设置与左右相关的值才起作用，比如 left、right；当父容器线性设置为 horizontal 横向时，只有设置与上下相关的值才起作用，比如 top、bottom。
- android:gravity 设置控件上面的内容在该组件里面的对齐方式。
- android:visibility 默认为 visibility，表示显示；设置为 invisibility 不显示，但是还会占据位置，留一个空白区域；设置成 gone 表示真正的完全隐藏。

3.2.2 使用代码控制线性布局管理器

正如 3.1 节所讲，在 Android 中所有的组件都是 android.view.View 类的子类，LinearLayout 类也不例外。对于这些 android.view.View 类的组件，除了使用配置文件的形式进行布局管理器的定义之外，还可以使用 Java 代码来动态控制。Android.widget.LinearLayout 类的重要操作方法和常量如表 3-1 所示。

表3-1 LinearLayout类的重要方法和常量

No.	方法及常量	类　型	描　述
1	public static final int HORIZONTAL	常量	设置水平对齐
2	public static final int VERTICAL	常量	设置垂直对齐
3	public LinearLayout(Context context)	构造	创建 LinearLayout 类的对象
4	public void addView(View child, ViewGroup, LayoutParams params)	普通	增加组件并指定布局参数
5	public void addView(View child)	普通	增加组件
6	public void onDraw(Canvas convas)	普通	用于图形绘制的方法
7	public void setOrientation(int orientation)	普通	设置对齐方式

另外，如果要使用程序控制 LinearLayout 布局管理器的操作，还需要对一些布局参数进行配置，这些参数都保存在 ViewGroup.LayoutParams 类中，线性布局的参数保存在 ViewGroup. LayoutParams 类的子类 LinearLayout.LayoutParams 类中。LinearLayout.LayoutParams 类的结构图如图 3-3 所示。

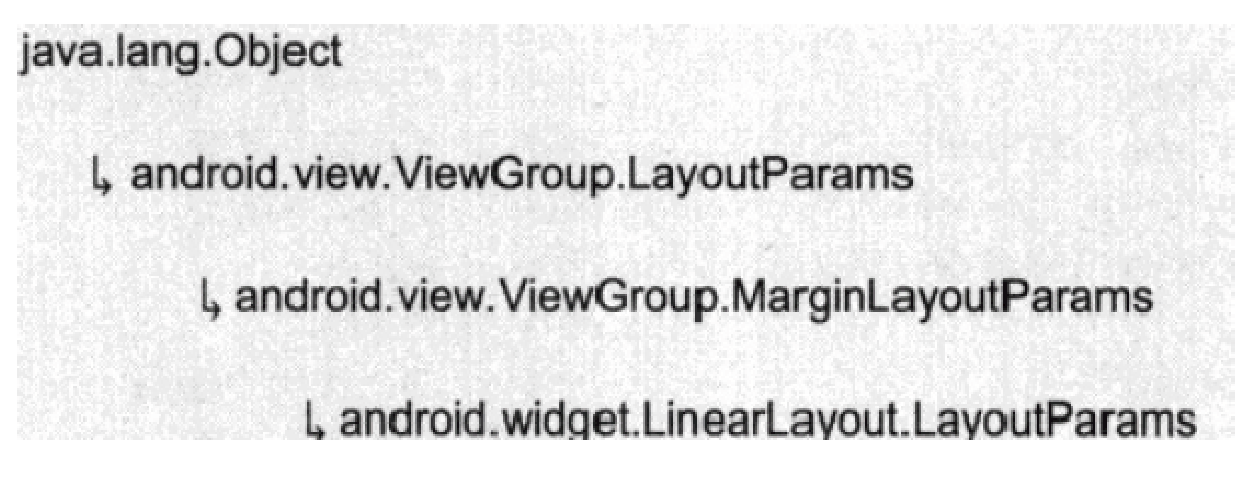


图 3-3 LinearLayout.LayoutParams 类的结构图

LinearLayout.LayoutParams 类提供了一个构造方法，具体如下：

```
public LinearLayout.LayoutParams (int width, int height)
```

在创建布局参数时需要传递布局参数的宽度和高度，而这两个布局参数可以通过 ViewGroup.LayoutParams 类提供的 FILL_PARENT（充满父元素）和 WRAP_CONTENT（包裹自身内容）两个常量参数来控制。

通过代码生成、控制布局管理器的代码实例如下：

```
package com.buaa.layout;

import android.app.Activity;
import android.os.Bundle;
import android.view.ViewGroup;
import android.widget.LinearLayout;
import android.widget.TextView;

public class ProductLinearLayoutActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        LinearLayout linearLayout = new LinearLayout(this);          //创建线性布局管理器
        LinearLayout.LayoutParams params = new LinearLayout.LayoutParams(
                ViewGroup.LayoutParams.MATCH_PARENT,                 //设置宽度为充满父元素
                ViewGroup.LayoutParams.MATCH_PARENT);                //设置高度为充满父元素
        linearLayout.setOrientation(LinearLayout.VERTICAL);          //垂直布局

        TextView textView = new TextView(this);                      //创建一个文本控件
        LinearLayout.LayoutParams textParams = new LinearLayout.LayoutParams(
                ViewGroup.LayoutParams.WRAP_CONTENT,                 //设置宽度为包裹内容
                ViewGroup.LayoutParams.WRAP_CONTENT);                //设置高度为包裹内容
        textView.setLayoutParams(textParams);                        //将参数传入文本控件
        textView.setText("我是这本书的作者李瑞奇");                   //给文本控件传入内容
        textView.setTextSize(30);                                    //设置文字的大小

        linearLayout.addView(textView);                        //添加新的控件进入布局
        super.addContentView(linearLayout, params);            //activity 增加了要显示的组件和参数
    }
}
```

这个程序通过 Java 代码直接控制线性布局管理器和它的子元素，最终又通过 addContentView 方法使线性布局管理器在这个 Activity 中展示出来。程序实现的效果如图 3-4 所示。

图 3-4　代码控制线性布局管理器效果

在上面的程序中使用了 addContentView(View,LayoutParams)方法。这是 Activity 类向 Activity 中增加 View 的方法。之前，我们会发现每次使用一个 Activity 都会默认调用 setContentView(View)方法。setContentView(View)方法也是设置 Activity View 的方法。那么两个方法有什么区别呢？

两者的区别主要包括两点：

- 在此之前已添加的 UI 组件是否被移除。setContentView(View)会导致先前添加的被移除，即替换性的；而 addContentView(View,LayoutParams)不会移除先前添加的 UI 组件，即累积性的。
- 是否控制布局参数。addContentView(View,LayoutParams) 有两个参数，可以控制布局参数；setContentView(View)没有接收布局参数，默认使用 MATCH_PARENT，不过 setContentView(View)也有带两个参数的版本，可以控制布局参数，这里不再讲解。

3.3 TableLayout：表格布局管理器

表格布局管理器继承自 LinearLayout 线性布局管理器，用行、列方式来管理容器内的控件，表格布局不需要指定多少行列，布局内每添加一行 TableRow 表示添加一行，然后在 TableRow 添加子控件，容器的列数由包含列数最多的行决定。

3.3.1 TableLayout 实例与属性详解

TableLayout 布局文件实例：

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:collapseColumns="2"      //指定第 3 列不显示
    android:shrinkColumns="1"        //指定第 2 列可伸缩
    android:gravity="center"
    android:layout_gravity="center"
    android:stretchColumns="0">
    <TableRow>
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="1 行 1 列" />
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="1 行 2 列" />
```

```
            <TextView
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="1 行 3 列" />
        </TableRow>
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="hello liruiqi" />
        <!-- android:layout_column 属性指定该组件到该行的指定列，此处指定占据第二列-->
        <TableRow>
            <TextView
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_column="1"
                android:text="hello liruiqi" />
        </TableRow>
        <!-- layout_span 属性指定该组件占据多少列，此处指定占据两列-->
        <TableRow>
            <TextView
                android:layout_span="2"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="hello liruiqi" />
        </TableRow>
    </TableLayout>
```

程序中定义了一个 4 行 2 列的表格，运行效果如图 3-5 所示。

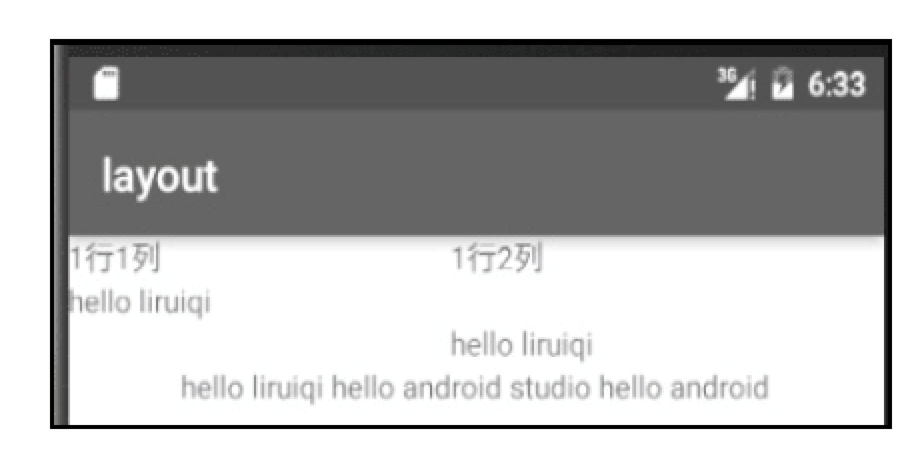


图 3-5 表格布局效果图

从上述 TableLayout 布局文件实例中可以清晰地看出，TableLayout 内部的子元素是按照表格来布局的，效果也达到了我们的预期。第 2 行只设置了一列，则只显示一列，第 3 行设置了 1 列，指定为第 2 列，第 4 行设置了一列内容，指定占据两列的控件，这些都正确无误地实现了，说明这些属性是可以起作用的。下面我们就布局文件中包含的一些常用属性做一些分析：

- android:collapseColumns 指定某一列不显示。
- android:layout_width 设置当前组件的宽度，match_parent 表示充满整个父元素，若使用 wrap_content 则意味着组件多大就多大。
- android:layout_height 设置当前组件的高度，match_parent 表示充满整个父元素，若使用 wrap_content 则意味着组件多大就多大。

- android:visibility　默认为 visibility，表示显示；设置为 invisibility 不显示，但是还要占据位置，留一个空白区域；设置成 gone 表示真正的完全隐藏。
- android:stretchColumns　为 TableLayout 容器设置属性，表示被设置的这些列可拉伸（注意：TableLayout 中列的索引从 0 开始）。
- android:shrinkColumns　为 TableLayout 容器设置属性，表示被设置的这些列可收缩。
- android:layout_column　为容器里面的控件设置属性，指定控件在 TableRow 中指定列。
- android:layout_span　为容器里面的控件设置属性，指定控件在 TableRow 中的指定列的数量。

3.3.2　使用代码控制表格布局管理器

TableLayout 是 LinearLayout 类的子类。与 LinearLayout 一样，TableLayout 也可以用 Java 代码来动态生成、控制布局管理器。与线性布局管理器类似，Android 提供了 Android.widget.TableLayout 和 Android.widget.TableRow 两个布局管理类，以及 Android.widget.TableLayout.LayoutParams 和 Android.widget.TableRow.LayoutParams 两个布局参数类来实现 Java 代码操作布局管理器。

通过代码生成、控制布局管理器的代码实例如下：

```
package com.buaa.layout;

import android.app.Activity;
import android.os.Bundle;
import android.view.ViewGroup;
import android.widget.LinearLayout;
import android.widget.TableLayout;
import android.widget.TableRow;
import android.widget.TextView;

public class ProductTabLayoutActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        TableLayout tableLayout = new TableLayout(this);             //创建表格布局管理器
        TableLayout.LayoutParams params = new TableLayout.LayoutParams(
                ViewGroup.LayoutParams.MATCH_PARENT,                 //设置宽度为充满父元素
                ViewGroup.LayoutParams.MATCH_PARENT);                //设置高度为充满父元素
        for (int i = 0; i < 5; i++) {
            TableRow tableRow = new TableRow(this);                  //创建一列
            for (int j = 0; j < 4; j++) {
                TextView textView = new TextView(this);
                textView.setText("第"+i + "行第" + j + "列");
                tableRow.addView(textView);                          //将文本加入此列
            }
```

```
                tableLayout.addView(tableRow);                     //将一列加入表格布局管理器
            }
            tableLayout.setStretchAllColumns(true);                //设置每一列都可扩展
            super.addContentView(tableLayout, params);             //将布局展示出来
        }
    }
```

这个程序通过 Java 代码动态生成表格布局管理器，并通过循环方式生成 TableRow 和 TextView，最终又通过 addContentView 方法使布局管理器在这个 Activity 中展示出来。程序实现的效果如图 3-6 所示。

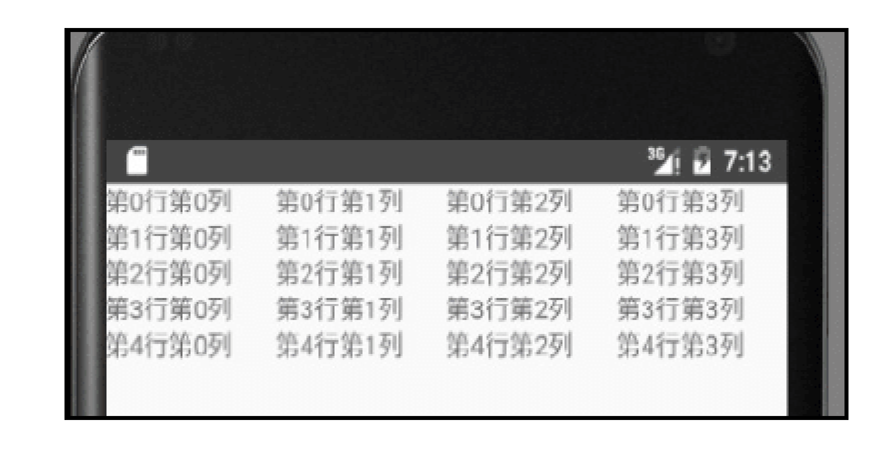


图 3-6　代码控制表格布局效果图

3.4　RelativeLayout：相对布局管理器

相对布局管理器内的控件布局总是相对于父容器或兄弟组件的位置，相对布局是实际中应用最多、最灵活的布局管理器。

3.4.1　RelativeLayout 实例及属性详解

RelativeLayout 布局文件实例：

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/relative"                  //设置 id，给下面的代码控制实例使用
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        android:id="@+id/txtInfor"              //设置 id
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="请输入短信内容"
        android:textSize="30sp" />

    <EditText
        android:id="@+id/txtSMS"                //设置 id
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_below="@id/txtInfor"     //设置位置在 id 为 textInfor 的控件下面
```

```
            android:background="#00eeff"                    //设置背景色
            android:minHeight="100dp" />

        <Button
            android:id="@+id/btnClearSMS"                   //设置 id
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_alignParentRight="true"
            android:layout_below="@id/txtSMS"               //设置位置在 id 为 txtSMS 的控件下方
            android:layout_marginRight="80dp"               //设置距离右边 80dp
            android:text="清除" />

        <Button
            android:id="@+id/btnSendSMS"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_alignBaseline="@id/btnClearSMS"      //与 id 为 txtSMS 的控件同水平线
            android:layout_alignParentRight="true"
            android:layout_marginRight="10dp"
            android:text="发送" />
    </RelativeLayout>
```

在这个程序中使用相对布局，包括一个 TextView 控件、EditText 控件和两个 button 控件。对于 EditText 控件和 button 控件，读者现在可能还不是很熟悉，暂时只需要这样使用即可，之后会有详细的讲解。使用了相对布局之后，内部控件会按照与其他控件的相对位置来布局。程序运行效果如图 3-7 所示。



图 3-7　相对布局效果图

从上述 RelativeLayout 布局文件实例中可以清晰地看出，RelativeLayout 内部的子元素是相对其他子元素来布局的。在上述例子中我们展示了一部分 RelativeLayout 的属性，下面再具体介绍下 RelativeLayout 其他的一些重要属性（见表 3-2）。

表3-2　Relativelayout重要属性

属　　性	描　　述
android:layout_above	将该控件的底部置于给定 id 的控件之上
android:layout_below	将该控件的底部置于给定 id 的控件之下
android:layout_toLeftOf	将该控件的右边缘与给定 id 的控件左边缘对齐
android:layout_toRightOf	将该控件的左边缘与给定 id 的控件右边缘对齐
android:layout_alignBaseline	将该控件的 baseline 与给定 id 的 baseline 对齐
android:layout_alignTop	将该控件的顶部边缘与给定 id 的顶部边缘对齐

（续表）

属　　性	描　　述
android:layout_alignBottom	将该控件的底部边缘与给定 id 的底部边缘对齐
android:layout_alignLeft	将该控件的左边缘与给定 id 的左边缘对齐
android:layout_alignRight	将该控件的右边缘与给定 id 的右边缘对齐
android:layout_alignParentTop	如果为 true,将该控件的顶部与其父控件顶部对齐
android:layout_alignParentBottom	如果为 true,将该控件的底部与其父控件底部对齐
android:layout_alignParentLeft	如果为 true,将该控件的左部与其父控件左部对齐
android:layout_alignParentRight	如果为 true,将该控件的右部与其父控件右部对齐
android:layout_centerInParent	如果为 true,将该控件置于父控件的中央
android:layout_centerVertical	如果为 true,将该控件置于垂直居中
android:layout_centerhorizontal	如果为 true,将该控件水平居中

3.4.2　使用代码控制相对布局管理器

与线性布局一样，相对布局也可以通过 Android.widget.RelativeLayout 类来动态控制，所有参数都可以通过 Android.widget.RelativeLayout.RelativeLayout.LayoutParams 类来控制。由于相对布局必须以组件作为布局参考，因此相对布局管理器的代码控制是在上面的程序基础上做改动来进行的。代码实例如下：

```
package com.buaa.layout;

import android.app.Activity;
import android.os.Bundle;
import android.view.ViewGroup;
import android.widget.RelativeLayout;
import android.widget.TextView;

public class ProductRelativeActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.relative);                    //调用布局上面的文件

        RelativeLayout relativeLayout = (RelativeLayout) findViewById(R.id.relative);
        RelativeLayout.LayoutParams params = new RelativeLayout.LayoutParams(
                ViewGroup.LayoutParams.MATCH_PARENT,    //设置宽度为充满父元素
                ViewGroup.LayoutParams.MATCH_PARENT);   //设置高度为充满父元素
        params.addRule(RelativeLayout.BELOW,R.id.txtSMS);     //设置参数，在 id 为 txtSMS 的下方
        TextView textView = new TextView(this);              //创建一个 TextView
        textView.setText("已发送");
        textView.setTextSize(28);
```

```
            relativeLayout.addView(textView,params);
        //将这个 TextView 加入 RelativeLayout，位置由 params 参数决定
        }
    }
```

这个程序通过 Java 代码直接控制相对布局管理器及其子元素，成功地在 id 为 txtSMS 的 EditText 控件下面加入了一个 TextView 控件。程序实现的效果如图 3-8 所示。



图 3-8　代码控制的相对布局效果图

3.5　FrameLayout：帧布局管理器

帧布局管理器为容器内的控件创建一块空白区域（帧），一帧一个控件，后面添加的控件覆盖在前面的控件上面，类似于 Java AWT 中的 CardLayout 布局。例如，在播放器 App 中，播放器上面的按钮就浮动在播放器上面。

3.5.1　FrameLayout 布局实例

FrameLayout 布局文件实例：

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:width="280dp"               //设置控件宽度
        android:height="280dp"              //设置控件高度
        android:background="#eeff00" />     //设置背景色

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:width="180dp"
        android:height="180dp"
        android:background="#3322ff" />
```

```
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:width="80dp"
        android:height="80dp"
        android:background="#ff2233" />
</FrameLayout>
```

在这个程序中使用帧布局，包括 3 个 TextView 控件，并为不同控件设置了不同背景色。由于采用了帧布局，因此 3 个控件会集中到一个地方并重叠。程序运行效果如图 3-9 所示。

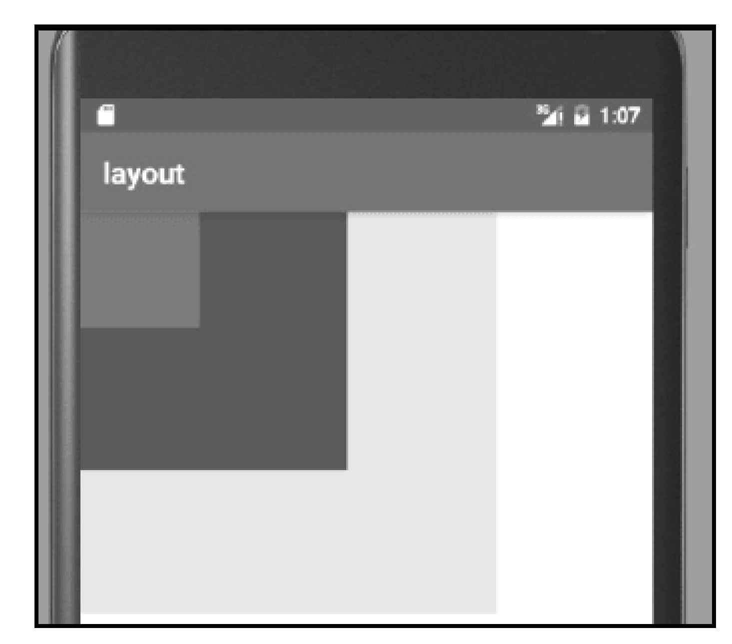


图 3-9　帧布局效果图

3.5.2　使用代码控制帧布局管理器

与前几种布局管理器一样，帧布局也可以通过 Android.widget.FrameLayout 类来动态控制，所有的参数也可以通过 Android.widget.FrameLayout.LayoutParams 类来控制。

通过 Android.widget.FrameLayout 类和 Android.widget.FrameLayout.LayoutParams 类控制帧布局的代码实例如下：

```
package com.buaa.layout;

import android.app.Activity;
import android.graphics.Color;
import android.os.Bundle;
import android.view.ViewGroup;
import android.widget.Button;
import android.widget.FrameLayout;
import android.widget.TextView;

public class ProductFrameActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        FrameLayout frameLayout = new FrameLayout(this);            //创建帧布局管理器
        FrameLayout.LayoutParams layoutParams = new FrameLayout.LayoutParams(
                ViewGroup.LayoutParams.MATCH_PARENT,            //设置宽度为充满父元素
                ViewGroup.LayoutParams.MATCH_PARENT);           //设置高度为充满父元素


        FrameLayout.LayoutParams params = new FrameLayout.LayoutParams(
                ViewGroup.LayoutParams.WRAP_CONTENT,            //设置宽度为包裹内容
```

```
                ViewGroup.LayoutParams.WRAP_CONTENT);       //设置高度为包裹内容
        TextView textView = new TextView(this);             //创建一个文本控件
        textView.setLayoutParams(params);                   //将参数传入文本控件
        textView.setText("这是帧布局的一个文本");

        Button button = new Button(this);
        button.setLayoutParams(params);
        button.setText("李瑞奇");

        frameLayout.addView(textView);
        frameLayout.addView(button);
        super.addContentView(frameLayout, layoutParams);    //设置要显示的组件和参数
    }
}
```

这个程序不使用布局管理器文件对组件进行配置，而是直接在 Activity 中完成这些操作，先定义一个帧布局，再创建几个控件，并加入帧布局中。程序实现的效果如图 3-10 所示。

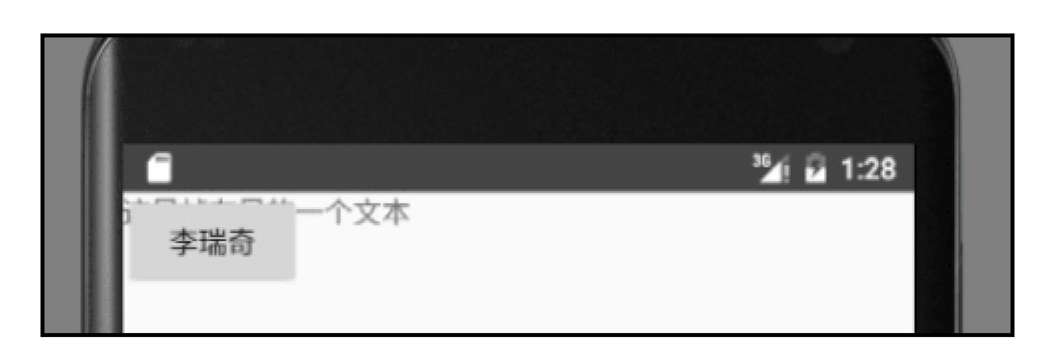


图 3-10　代码控制的帧布局效果图

3.6　AbsoluteLayout：绝对布局管理器

本节所讲解的绝对布局管理器由于版本的升级原因，在 Android 2.3.3 中已经被表面定义为不建议使用，考虑到不同 Android 版本的开发者，以及绝对布局管理器在一定情况下也还在使用，在此就简单介绍一下。绝对布局管理容器内部控件的位置以及大小需要开发人员通过指定 X、Y 坐标来定义。

AbsoluteLayout 布局文件实例：

```
<?xml version="1.0" encoding="utf-8"?>
<AbsoluteLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:height="100dp"
        android:width="100dp"
```

```
            android:layout_x="200dp"     //指定的横坐标
            android:layout_y="20dp"      //指定的纵坐标
            android:textSize="20sp"
            android:text="这是中文的绝对布局"
            />
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:height="500dp"
            android:width="200dp"
            android:layout_x="10dp"
            android:layout_y="250dp"
            android:textSize="20sp"
            android:text="this is a absolutelayout"/>
    </AbsoluteLayout>
```

在这个程序中使用绝对布局把两个 TextView 控件放到了指定位置上。程序的运行效果如图 3-11 所示。

看到这可能有些读者会想，绝对布局看起来还是很有用的，为什么会被废弃掉呢？原因是绝对布局需要指定绝对的坐标值，在开发中我们会经常改变组件大小，就会使得对显示的控制变复杂。而且使用绝对布局会导致不同机型上的显示不一致。所以，这里不过多讲解绝对布局，也希望读者在开发时尽量不要使用绝对布局。当然有废弃也会有新增，3.7 节将会讲解一个新增加的布局管理器。



图 3-11 绝对布局效果图

3.7 GridLayout：网格布局管理器

网格布局管理器是 Android 4.0 以后新增加的布局管理器。网格布局管理器将容器划分为行×列的网格，每个控件置于网格中，当然也可以通过设置相关属性使一个控件占据多行或多列。

3.7.1 GridLayout 实例及属性详解

GridLayout 相比其他的布局管理器的常用属性如表 3-3 所示。

表3-3 GridLayout常用属性

属 性	说 明
android:columnCount="4"	设置网格布局有 4 列
android:rowCount="4"	设置网格布局有 4 行

（续表）

属　　性	说　　明
android:layout_row = "1"	设置组件位于第 2 行
android:layout_column = "2"	设置该组件位于第 3 列
android:layout_rowSpan = "2"	设置纵向横跨 2 行
android:layout_columnSpan = "3"	设置横向横跨 3 列

布局文件实例如下：

```
<GridLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:rowCount="6"
    android:columnCount="4"
    android:orientation="horizontal">

    <TextView
        android:layout_columnSpan="4"
        android:text="0"
        android:textSize="50sp"
        android:layout_marginLeft="5dp"
        android:layout_marginRight="5dp" />
    <Button
        android:text="回退"
        android:layout_columnSpan="2"
        android:layout_gravity="fill" />
    <Button
        android:text="清空"
        android:layout_columnSpan="2"
        android:layout_gravity="fill" />
    <Button
        android:text="+" />
    <Button
        android:text="1" />
    <Button
        android:text="2" />
    <Button
        android:text="3" />
    <Button
        android:text="-" />
    <Button
        android:text="4" />
    <Button
```

```
            android:text="5" />
        <Button
            android:text="6" />
        <Button
            android:text="*" />
        <Button
            android:text="7" />
        <Button
            android:text="8" />
        <Button
            android:text="9" />
        <Button
            android:text="/" />
        <Button
            android:layout_width="wrap_content"
            android:text="." />
        <Button
            android:text="0" />
        <Button
            android:text="=" />
    </GridLayout>
```

在这个程序中使用了网格布局，用 TextView 和 Button 控件制作了一个简单计算器的布局。程序中通过 android:layout_rowSpan 和 android:layout_columnSpan 设置表明组件横跨的行数与列数，再通过:android:layout_gravity = "fill" 设置表明组件填满所横跨的整行或者整列。程序运行效果如图 3-12 所示。



图 3-12　网格布局效果图

3.7.2　使用代码控制网格布局管理器

与前几种布局管理器一样，网格布局也可以通过 Android.widget.GridLayout 类来动态控制，所有的参数也可以通过 Android.widget.GridLayout.LayoutParams 类来控制。

通过 Android.widget.GridLayout 类和 Android.widget.GridLayout.LayoutParams 类控制网格布局的代码实例如下：

```
package com.buaa.layout;

import android.app.Activity;
import android.os.Bundle;
import android.view.Gravity;
import android.view.ViewGroup;
import android.widget.Button;
import android.widget.GridLayout;
```

```
import android.widget.TextView;

public class ProductGridAcitivity extends Activity {
    String[] btnLable = new String[]{
            "7", "8", "9", "+",
            "4", "5", "6", "-",
            "1", "2", "3", "*",
            "0", ".", "=", "/"
    };

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        GridLayout gridLayout = new GridLayout(this);

        TextView textView = new TextView(this);      //创建一个文本控件
        textView.setText("0");
        textView.setTextSize(50);
        gridLayout.addView(textView);

        for (int i = 0; i < btnLable.length; i++) {      //添加 16 个网格，每个网格 1 个按钮

            Button button = new Button(this);
            button.setText(btnLable[i]);
            button.setTextSize(30);
            GridLayout.Spec rowSpec = GridLayout.spec(i / 4 + 2);          //行位置
            GridLayout.Spec colSpec = GridLayout.spec(i % 4);              //列位置
            GridLayout.LayoutParams gridLayoutParams = new
            GridLayout.LayoutParams(rowSpec, colSpec);   //将属性传入
            gridLayoutParams.setGravity(Gravity.FILL_HORIZONTAL);     //横向排满容器
            gridLayout.addView(button, gridLayoutParams);             //将按钮添加到网格容器内
        }
        GridLayout.LayoutParams layoutParams = new GridLayout.LayoutParams();
        super.addContentView(gridLayout, layoutParams);
    }
}
```

这个程序通过在 Activity 中使用 Java 代码动态操作布局文件的方式定义了网格布局，实现了和使用布局文件同样的效果。程序实现的效果如图 3-13 所示。

可以发现，这个效果和之前的效果是完全一样的。



图 3-13　代码控制的网格布局效果图

3.8　布局管理器之间互相嵌套

在使用布局管理器进行布局时会发现，有时候实际的需求不是一种布局管理器能够满足的，这时我们可以将多个布局管理器嵌套使用。用法和单个布局管理器的使用并无多大区别，这里就以 LinearLayout、GridLayout、RelativeLayout 三者的互相嵌套为例，实现一个带标题的计算器的布局。其他嵌套情况读者可根据实例随意变换，此处不做过多叙述。

LinearLayout、GridLayout、RelativeLayout 三者互相嵌套的布局文件实例：

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    //在相对布局内部嵌入一个线性布局
    <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
        android:id="@+id/linear"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical">
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="hello liruiqi1" />
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="hello liruiqi2" />
    </LinearLayout>
    //在相对布局内部嵌入一个网格布局，并将网格布局放在线性布局下方
    <GridLayout xmlns:android="http://schemas.android.com/apk/res/android"
```

```
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_below="@+id/linear"
            android:columnCount="4"
            android:orientation="horizontal"
            android:rowCount="6">
            <TextView
                android:layout_columnSpan="4"
                android:layout_marginLeft="5dp"
                android:layout_marginRight="5dp"
                android:text="0"
                android:textSize="50sp" />
            <Button
                android:layout_columnSpan="2"
                android:layout_gravity="fill"
                android:text="回退" />
            <Button
                android:layout_columnSpan="2"
                android:layout_gravity="fill"
                android:text="清空" />
            <Button android:text="+" />
            <Button android:text="1" />
            <Button android:text="2" />
            <Button android:text="3" />
            <Button android:text="-" />
            <Button android:text="4" />
            <Button android:text="5" />
            <Button android:text="6" />
            <Button android:text="*" />
            <Button android:text="7" />
            <Button android:text="8" />
            <Button android:text="9" />
            <Button android:text="/" />
            <Button
                android:layout_width="wrap_content"
                android:text="." />
            <Button android:text="0" />
            <Button android:text="=" />
        </GridLayout>
    </RelativeLayout>
```

本程序在相对布局内部嵌入了一个线性布局和一个网格布局，并将网格布局放在线性布局的下方。程序运行效果如图 3-14 所示。



图 3-14 嵌套布局效果图

这里在最外层使用一个 RelativeLayout，在 RelativeLayout 内部使用 LinearLayout 和 GridLayout。其中 LinearLayout 用来实现计算器的标题，即“hello liruiqi1”和“hello liruiqi2”，在实际开发中，可以替换成自己需要的内容。GridLayout 其实就是前面所讲述的计算器的布局。由于外层是 RelativeLayout，因此在确定位置时在 GridLayout 中使用了 layout_below 属性，使 GridLayout 落在 LinearLayout 下方。在 Android 开发中，嵌套使用布局管理器很常见，这里只是给出一个示例，读者可模仿操作并尝试使用其他布局管理器进行互相嵌套。

3.9 小　结

本章主要介绍了布局管理器的作用，并介绍了 Android 中的 6 种布局管理器，即 LinearLayout、RelativeLayout、TableLayout、FrameLayout、AbsoluteLayout、GridLayout。所有布局管理器都可以通过配置文件实现，也可以在 Activity 中用代码实现。布局管理器可以直接通过互相嵌套来实现更复杂的布局。



第 4 章 基本控件与事件处理

第 3 章在讲解布局管理器时介绍了整个 View 类的继承结构，读者会发现在 View 类的子类中大部分都是控件类。从 View 类图中也可以看出在讲解具体的布局时使用到的 TextView、Button 等也是基本控件。

在 Android 的图形界面（UI）的开发中有两个非常重要的内容：一个是控件的布局，另一个就是控件的事件处理。一个好的界面除了布局管理器之外还需要有基本控件，没有基本控件填充，再好的布局都是枉然。如果有了这些界面而没有事件处理，就变成了死界面。本章将讲解基本控件与控件的事件处理。

4.1 常用基本控件的使用

在 Android 开发中，需要使用的控件很多，除了之前提到过的 TextView、Button、EditText，还有 RadioGroup、CheckBox、Spinner、ImageView 等一大批控件。这些控件构成了 Android 图形界面开发的基石。同时，在使用这些控件时需要设置它们的宽与高，使用文字时需要设置自提点大小，这又将涉及 Android 中的尺寸问题。本节将重点讲解控件的使用，同时简单介绍 Android 的尺寸问题。

4.1.1 基本控件的使用

就像第 3 章所叙述的那样，Android 中的控件类都是 android.view.View 类的子类，都在 android.wegdit 包下，除了 TextView、Button 之外，还有很多控件类。总结起来，Android 中常用的控件类如表 4-1 所示。

表4-1 Android中常用的控件类

控件名称	描述	控件名称	描述
TextView	文本显示控件	SeekBar	拖动条控件
Button	按钮控件	ProgerssBar	进度条控件
EditText	文本编辑框控件	ScrollView	可滚动视图控件
ImageView	图片显示控件	DatePicker	日期显示控件
ImageButton	用图片作为按钮的控件	TimePicker	事件显示控件
RadioGroup	单选按钮控件	Dialog	对话框控件
CheckBox	复选框控件	Toast	信息提示框控件
Spinner	下拉列表控件		

通过第 3 章的讲解，读者应该已经明白了如何使用布局管理器，并明白了布局管理器在使用时需要配置很多属性，而这些属性是可以通过相对应的 Java 方法来操作的。同时第 3 章也简单介绍了如何使用一个控件，那就是直接将控件加入布局管理器中。除了这种方式外，还可以和布局管理器一样通过 Activity 程序来控制。同布局管理器一样，普通控件在使用时也需要配置很多属性，而这些属性也可以通过相对应的 Java 方法来操作。控件的常用属性很多，常用的却不多。同时不同的控件也有各自特有的属性，读者在使用过程中慢慢就能理解这些属性的意义了。控件中相同又最常用的属性还有几种，如表 4-2 所示。

表4-2 Android中常用的控件类

属性名称	操作方法名	描述
android：id	setId(int id)	设置控件 id
android：focusable	setFocusable(boolean focusable)	设置控件是否可以获得焦点

（续表）

属性名称	操作方法名	描　述
android：backgroud	setBackgroudResource(int res)	设置控件背景
android：visibility	setVisibility(int visibility)	设置控件是否可见

下面将通过实例来演示这些属性，在实例中还会涉及一些控件的特别属性。但是本书不会像其他书籍那样一下将所有的控件都讲解出来，这样会让刚刚接触 Android 的读者很难记忆。所以这里的实例将以 TextView、Button、EditText、ImageView、RadioGroup、SeekBar、Dialog、Toast 这几个最常用的控件为例，其余的控件会在之后的章节中通过实例一一展现，让读者在实例中慢慢理解。

1. TextView、Button、EditText、ImageView、RadioGroup、SeekBar 控件的使用

创建一个 Activity 类 ShowViewActivity，将对应的布局文件 activity_show_view.xml 修改如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:gravity="center"                        //让布局管理器内的控件居中排列
    android:orientation="vertical"
    tools:context="com.buaa.view.activity.ShowViewActivity">

    <TextView
        android:id="@+id/show_text"                 //为控件添加一个 id
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="本节只讲述一部分控件"          //为文本控件添加文件
        android:textColor="#FF4040"                 //为文字添加颜色
        android:textSize="24sp"                     //为控件文字设置字体大小
        android:visibility="visible" />             //设置控件为可见，默认为可见

   <EditText
        android:id="@+id/show_edit"
        android:layout_width="match_parent"
        android:layout_height="30dp"
        android:enabled="true"                      //将编辑框设置为可编辑，默认为可编辑
        android:hint="请输入文字"                   //设置编辑框提示文字
        android:inputType="number"                  //设置编辑框输入类型为数字，默认为文字
        android:textSize="24sp" />                  //为输入的文字设置大小
    <SeekBar                                        //拖拉控件，常在播放器应用中使用
        android:id="@+id/seek_bar"
```

```
        android:layout_width="match_parent"
        android:layout_height="30dp" />             //将控件高度设置为 30dp

    <Button
        android:id="@+id/show_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="展现一个 button 按钮"          //为按钮设置显示的文本
        android:visibility="gone" />                //将控件设置为不可见，同时不会占据空间

    <ImageView
        android:id="@+id/show_image"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@drawable/img"                 //为 ImageView 设置要显示的图片
        android:visibility="invisible" />           //将控件设置为不可见，会占据空间

    <RadioGroup
        android:id="@+id/show_group_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:orientation="horizontal">           //将控件内部的组件设置为不可见

        <RadioButton                                //与单选按钮配套的按钮
            android:id="@+id/lanqiu"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="篮球" />

        <RadioButton
            android:id="@+id/paiqiu"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="排球" />

        <RadioButton
            android:id="@+id/pingpang"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="乒乓" />
    </RadioGroup>

</LinearLayout>
```

在上述布局中用注释的方式给控件的一些属性做了解释。这里不再解释，运行工程，在模拟器上显示的界面如图 4-1 所示。



图 4-1　控件在布局中的显示效果

这里隐藏了图片和按钮，但是会发现一大块空白区域，这就是在设置 ImageView 不可见时使用的是 invisible，这种方式还会占据控件。通过测试发现 EditText 也确实是智能输入数字。下面在 Activity 类中创建一个 initView()方法，在 initView()方法中通过 findViewById(int id)方法获取相应的控件，最后在 onCreate()方法中调用 initView()方法。initView()代码如下：

```
private void initView() {
    //获取 TextView 控件
    TextView textView = (TextView) findViewById(R.id.show_text);
    //设置 TextView 控件的内容
    textView.setText("通过代码控制的 TextView");
    //设置 TextView 控件文字大小
    textView.setTextSize(20);

    //获取 EditText 控件
    EditText editText = (EditText) findViewById(R.id.show_edit);
    //获取 EditText 输入内容，此时 EditText 中没有内容，注释掉
    // editText.getText().toString();
    //设置 EditText 输入内容为 Text 类型
    editText.setInputType(InputType.TYPE_CLASS_TEXT);

    //获取 Button 控件
    Button button = (Button) findViewById(R.id.show_button);
    //设置为可见
    button.setVisibility(View.VISIBLE);

    //获取 ImageView 控件
```

```
        ImageView imageView = (ImageView) findViewById(R.id.show_image);
        //设置为可见
        imageView.setVisibility(View.VISIBLE);
        //设置 ImageView 的图片
        imageView.setImageResource(R.drawable.ic_launcher);

        //获取 RadioGroup 控件
        RadioGroup radioGroup = (RadioGroup) findViewById(R.id.show_group_button);
        //默认选中排球
        radioGroup.check(R.id.paiqiu);
    }
```

为了方便读者理解，在上述代码中做了很详细的注释。上述代码主要做了 5 件事：改变布局文件中的 TextView 文字；设置 EditText 的输入类型为 Text；将 Button 按钮设置为可见；将 ImageView 按钮设置为可见，并修改图片；将 id 为“paiqiu”的选项设置为 RadioGroup 的默认选项。运行程序，在模拟器上显示的界面如图 4-2 所示。



图 4-2　使用代码控制控件在布局中的显示

通过这样一个实例，读者应该能够使用上述几个控件了，想要进一步精通只能靠以后的实践去积累了。在之后的内容中我们还会频繁使用上述几个控件，但会涉及新的属性、新的方法。

2. Toast 控件的使用

Toast 是 Android 中用来显示信息的一种机制，没有焦点，过一定的时间就会自动消失。使用 Toast 很简单，只需要设置要显示的内容、显示时长、显示位置之后调用 show()方法就可以了。设置内容等的方式有两种，代码如下：

```
    private void toast() {
        //一般情况下使用 Toast 类的静态方法 makeText，然后调用 show()方法就可以了
        //第一个参数表示 context 对象，第二个为显示内容，第三个为显示长度
        Toast toast = Toast.makeText(this,"显示位置的方式",Toast.LENGTH_LONG);
        //设置 Toast 显示的位置，一般不调用此方法，会默认显示到界面底部
        toast.setGravity(Gravity.CENTER, 0, 0);
        toast.show();
        //一般情况下会使用下面这种方式显示 Toast
        Toast.makeText(this,"常用方式",Toast.LENGTH_SHORT).show();
    }
```

上述代码在 onCreate()方法中调用了 toast()方法。运行程序，在中部和底部会依次出现两个 Toast 提示框。如图 4-3 所示为先出现的指定位置的 Toast，图 4-4 所示为默认位置的 Toast。

其实，还有一种可以自定义 Toast 布局的方法来显示 Toast，只不过在实际开发中并不常用，这里仅给出代码，不做分析，读者如有兴趣可以自行尝试。




图 4-3　指定位置的 Toast



图 4-4　默认的 Toast

```
private void myToast(){
    //设置自定义的布局
    LayoutInflater inflater = getLayoutInflater();
    View layout = inflater.inflate(R.layout.custom,
            (ViewGroup) findViewById(R.id.my_toast));
    ImageView image = (ImageView) layout
            .findViewById(R.id.toast_icon);
    image.setImageResource(R.drawable.icon);
    TextView title = (TextView) layout.findViewById(R.id.toast_title);
    title.setText("自定义");
    TextView text = (TextView) layout.findViewById(R.id.toast_text);
    text.setText("自定义 Toast");
    Toast toast = new Toast(getApplicationContext());
    toast.setGravity(Gravity.RIGHT | Gravity.TOP, 12, 40);
    toast.setDuration(Toast.LENGTH_LONG);
    //将自定义的布局传入
    toast.setView(layout);
    toast.show();
}
```

3. Dialog 控件的使用

Dialog 控件在应用中是必不可少的一个组件，在 Android 中也不例外。Dialog 控件会提示一些重要信息，同时对一些需要用户额外交互的内容也很有帮助。一个 Dialog 就是一个小窗口，并不会填满整个屏幕，通常是以模态显示，要求用户必须采取行动才能继续进行剩下的操作。Android 中提供了丰富的对话框支持，通常包括如下 4 种常用的对话框：

- AlertDialog 警告对话框，是使用最广泛、功能最丰富的一个对话框。
- ProgressDialog 进度条对话框，只是对进度条进行了简单的封装。
- DatePickerDialog 日期对话框。
- TimePickerDialog 时间对话框。

所有的对话框都直接或间接继承自 Dialog 类，而 AlterDialog 直接继承自 Dialog，其他的几个类均继承自 AlterDialog。在实际开发中主要使用的是 AlertDialog 以及由 AlertDialog 自定义而来的对话框，所以本部分主要讲解 AlertDialog。

AlterDialog 可以包含一个标题、一个内容消息或者一个选择列表、最多 3 个按钮。推荐使用一个内部类 AlterDialog.Builder 来创建 AlterDialog。使用 Builder 对象可以设置 AlterDialog 的各种属性，再通过 Builder.create()就可以得到 AlterDialog 对象。如果只是需要显示 AlterDialog，一般可以直接使用 Builder.show()方法，返回一个 AlterDialog 对象，并且显示 AlterDialog。

如果仅仅是需要提示一段信息给用户，就可以直接使用 AlterDialog 的一些属性设置提示信息，涉及的方法有：

- AlterDialog create() 根据设置的属性创建一个 AlterDialog。
- AlterDialog show() 根据设置的属性创建一个 AlterDialog，并显示在屏幕上。
- AlterDialog.Builder setTitle() 设置标题。
- AlterDialog.Builder setIcon() 设置标题的图标。
- AlterDialog.Builder setMessage() 设置标题的内容。
- AlterDialog.Builder setCancelable() 设置是否模态，一般设置为 false，表示模态，要求用户必须采取行动才能继续进行剩下的操作。

下面通过一个实例来展示 AlertDialog 的使用。在 Activity 中创建一个 myDialog()方法，并在 onCreate()方法中调用它。代码如下：

```
private void myDialog() {
    //初始化一个 AlertDialog 的内置对象 builder
    AlertDialog.Builder builder = new AlertDialog.Builder(
            ShowViewActivity.this);
    //设置 Dialog 的 title
    builder.setTitle("提示");
    //设置显示内容
    builder.setMessage("正在显示的是包含多个按钮以及一个 icon 为美女的对话框");
    //设置 icon
    builder.setIcon(R.drawable.img);

    //添加一个确定的按钮
    builder.setPositiveButton("确定", new DialogInterface.OnClickListener() {
        //设置当点击确定后使用 Toast 显示一行文字，并让 Dialog 消失
        @Override
        public void onClick(DialogInterface dialog, int which) {
```

```
                Toast.makeText(ShowViewActivity.this, "确定被点击",
                        Toast.LENGTH_SHORT).show();
                dialog.dismiss();
            }
        });
        //添加一个否定以后再说按钮
        builder.setNegativeButton("以后再说", new DialogInterface.OnClickListener() {
            //设置当点击此按钮后使用 Toast 显示一行文字，并让 Dialog 消失
            @Override
            public void onClick(DialogInterface dialog, int which) {
                // TODO Auto-generated method stub
                Toast.makeText(ShowViewActivity.this, "以后再说被点击",
                        Toast.LENGTH_SHORT).show();
                dialog.dismiss();
            }
        });
        //添加一个忽略按钮
        builder.setNeutralButton("忽略", new DialogInterface.OnClickListener() {
            //设置当点击按钮后使用 Toast 显示一行文字，并让 Dialog 消失
            @Override
            public void onClick(DialogInterface dialog, int which) {
                // TODO Auto-generated method stub
                Toast.makeText(ShowViewActivity.this, "忽略被点击",
                        Toast.LENGTH_SHORT).show();
                dialog.cancel();
            }
        });
        //使用模态，即不对上述 3 个按钮做操作就不能继续其他操作。默认为可以继续进行其他操作
        builder.setCancelable(false);
        //调用 show()方法显示这个 Dialog
        builder.show();
    }
```

由于在代码中做了很好的注释，因此对代码内容不再做更多解释。运行工程，自定义 Dialog 的效果如图 4-5 所示，自定义 Toast 的效果如图 4-6 所示。



图 4-5　默认的 Dialog



图 4-6　默认的 Toast

4.1.2 Android 中的尺寸问题

通过前一部分的学习，可以发现不管是字体设置还是控件宽高的设置都需要尺寸，尺寸的设置会极大地影响 UI 界面的开发质量。

过去，程序员通常以像素为单位设计计算机用户界面，例如图片大小为 80×32（像素）。这样处理的问题在于，如果在一个每英寸点数（dpi）更高的新显示器上运行该程序，那么用户界面会显得很小。在有些情况下，用户界面可能会小到难以看清内容。为了解决这个问题，Android 中采用了与分辨率无关的度量单位来开发程序。Android 应用开发支持不同的度量单位，常用的尺寸主要有 px、dp、sp。

（1）px，即像素，1px 代表屏幕上一个物理像素点。在 Android 中 px 单位不被建议使用，因为同样像素的图片在不同手机上显示的实际大小可能不同。

（2）dp，这是最常用但也最难理解的尺寸单位。它与“像素密度”密切相关，所以这里首先解释一下什么是像素密度。假设有一部手机，屏幕的物理尺寸为 1.5 英寸×2 英寸，屏幕分辨率为 240×320，就可以计算出在这部手机屏幕上每英寸包含的像素点的数量为 240/1.5=160dpi（横向）或 320/2=160dpi（纵向），160dpi 就是这部手机的像素密度。像素密度的单位 dpi 是 Dots Per Inch 的缩写，即每英寸像素数量。横向和纵向的值都是相同的，因为大部分手机屏幕使用正方形的像素点。

不同的手机/平板可能具有不同的像素密度，例如同为 4 寸手机，有 480×320 分辨率的，也有 800×480 分辨率的，前者的像素密度就比较低。Android 系统定义了 4 种像素密度：低（120dpi）、中（160dpi）、高（240dpi）和超高（320dpi），对应的 dp 到 px 的系数分别为 0.75、1、1.5 和 2，这个系数乘以 dp 长度就是像素数。例如，界面上有一个长度为 80dp 的图片，那么它在 240dpi 的手机上实际显示为 80×1.5=120px，在 320dpi 的手机上实际显示为 80×2=160px。将这两部手机放在一起对比，就会发现这个图片的物理尺寸“差不多”。

（3）sp，与缩放无关的抽象像素（Scale-independent Pixel）。sp 和 dp 类似，唯一的区别就是 Android 系统允许用户自定义文字尺寸大小（小、正常、大、超大等）。当文字尺寸是“正常”时 1sp=1dp=0.00625 英寸，而当文字尺寸是“大”“或”“超大”时 1sp>1dp=0.00625 英寸，类似于我们在 Windows 里调整字体尺寸以后的效果：窗口大小不变，只有文字大小改变。

在经过长时间的开发实践后，最终总结出了使用这几种尺寸的规律：文字的尺寸一律用 sp 单位，非文字的尺寸一律使用 dp 单位，只有在一些特殊时候才会使用 px 单位，如需要在屏幕上画一条细的分隔线时。

4.2 Android 中的事件处理

在 Android 的图形界面（UI）开发中，有两个非常重要的内容：一个是控件的布局，另一个就是控件的事件处理。在 4.1 节讲解了常用的控件，本节将主要讲解事件的处理。Android 中的常用事件有点击事件、长按事件、触摸事件、焦点事件、按键事件、下拉列表的选中事件、单选按钮的改变事件等。对于事件的处理，基本上可以总结为 3 个步骤：

步骤 01 获取触发事件的对象，比如点击了一个 Button，如果要对这个点击事件进行处理，就需要获取 Button 的对象。

步骤 02 实现一个对应的事件处理接口。每个事件都有对应的事件处理接口，在事件处理中必须要实现事件处理接口，同时要实现其中的事件处理方法。在一个事件处理接口的实现类中可以处理多个事件。

步骤 03 用获取的控件对象调用该控件的某个事件监听方法，将第二步实现的接口类的对象作为参数传入，并对该事件进行注册。

下面按照上述步骤，根据不同的事件来进行详细讲解。

4.2.1 点击事件

点击事件，顾名思义就是点击了某个控件而触发的事件。点击事件常见于 Button 按钮，当然，TextView、ImageView 等控件中也有使用。另外，布局管理器（如 LinearLayout 等）也是可以有点击事件的。下面将以 Button 与 TextView 为实例来进行讲解。

Activity 对应的布局文件代码：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context="com.buaa.view.activity.ClickActivity">

    <Button
        android:id="@+id/click_event"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="点击我进行测试" />

    <TextView
        android:id="@+id/text_event"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="我是 TextView"
        android:textSize="26sp"/>
</LinearLayout>
```

Activity 中处理的代码：

```
package com.buaa.view.activity;

import android.support.v7.app.AppCompatActivity;
```

```
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;
import android.widget.Toast;

import com.buaa.view.R;

public class ClickActivity extends AppCompatActivity implements View.OnClickListener {

    private Button button;
    private TextView textView;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_click);

        intiView();
    }

    private void intiView() {
        button = (Button) findViewById(R.id.click_event);
        textView = (TextView) findViewById(R.id.text_event);

        button.setOnClickListener(this);
        textView.setOnClickListener(this);
    }

    @Override
    public void onClick(View v) {
        switch (v.getId()){
            case R.id.text_event:
                Toast.makeText(this,"您点击了一个 TextView",Toast.LENGTH_LONG).show();
                break;
            case R.id.click_event:
                Toast.makeText(this,"您点击了一个 Button",Toast.LENGTH_LONG).show();
                break;
        }
    }
}
```

就像前文所说的分 3 步来处理事件：第一步使用 findViewById()方法获取 Button 按钮的对象；第二步让 Activity 实现点击事件的处理接口 View.OnClickListener，并实现 onClick()方法，在此方法中响应具体的点击事件；第三步调用 setOnClickListener()点击事件的监听方法，并将 View.OnClickListener 接口实现类的对象传入（实例中是用 Activity 实现的，所以传入了 this 对象）。

对点击事件的处理都是要在 onClick()方法中进行编写的，实例在点击之后弹出一个 Toast 进行提示。如果只有一个点击事件，就可以直接在方法内编写响应程序，但是当实例中有两个（或多个）需要进行处理的点击事件时则需要使用一个 switch 根据它们的 id 进行判断。在第三步调用 setOnClickListener()方法时，系统会对 View 进行注册，所以在 onClick()方法中可以从 view.getId()方法获取对应的 View 控件的 id。

运行程序，点击 Button 按钮的效果如图 4-7 所示，点击 TextView 的效果如图 4-8 所示。




图 4-7　点击 Button 的效果图



图 4-8　点击 TextView 的效果图

其实，在处理点击事件时，实现事件处理接口的方式有 3 种，实例中的处理方式是在实际开发中最常用的。下面简单介绍其他两种方式。

（1）匿名内部类的方式

直接在控件对象调用 setOnClickListener()方法时以匿名内部类的方式传入事件处理接口的对象。看下面这个例子：

```
button.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Toast.makeText(ClickActivity.this,"您点击了一个 Button",Toast.LENGTH_LONG).show();
    }
});
```

使用这种方式就需要在 Button 按钮和 TextView 中都这样处理。如果不是两个，同时需要

的点击事件特别多，就会使得代码冗余，让 initView()方法过于庞大，同时可读性降低。选择使用实例中的方法就会显得很清晰，同时代码简洁、可读。当然，如果需要处理的事件只有一个或者几个的话，使用此种方式会相当简单。只是在真正的开发中只有一个事件需要处理的情况相对较少，所以实例中的处理方式才是应用最广泛的。

（2）内部类的方式

这种方式和匿名内部类方式不同的是，它选择在 Activity 内部创建一个内部类，并在内部类内实现 onClick(View view)方法。内部类如下：

```
private class clickListener implements View.OnClickListener {
    @Override
    public void onClick(View v) {
        switch (v.getId()) {
            case R.id.text_event:
                Toast.makeText(ClickActivity.this, "您点击了一个 TextView",
Toast.LENGTH_LONG).show();
                break;
            case R.id.click_event:
                Toast.makeText(ClickActivity.this, "您点击了一个 Button",
Toast.LENGTH_LONG).show();
                break;
        }
    }
}
```

此时，调用 setOnClickListener()方法的方式如下：

```
button.setOnClickListener(new clickListener());
textView.setOnClickListener(new clickListener());
```

使用这种方式，和实例中的方式并无区别，只是一个用 Activity 实现了事件处理接口，一个使用了内部类的方式。当然，也可以用普通类的方式，和内部类效果一样，但是在代码的内聚性上会大打折扣。当然如果遇到的工程相对复杂，那么为了解耦有时也会使用普通类的方式，不过非常少见。

只有理解了上面 3 种实现事件处理接口的方式，在开发中才能根据需要选择不同的方式。其实，不只是点击事件，其他事件实现事件处理接口也是同样的。在下面的讲解中主要使用第一种方式。

4.2.2 长按事件

长按事件就是长按了某个控件而触发的事件。TextView、ImageView、Button 等控件经常会使用长按事件。另外，布局管理器（如 LinearLayout）也是可以有长按事件的。下面将以 LinearLayout 为实例来进行讲解。实例在点击事件的基础上进行如下修改即可：

（1）给 LinearLayout 添加 id。在布局 LinearLayout 中加入：

```
android:id="@+id/long_click_event"
```

（2）让 Activity 实现长按事件的事件处理接口 View.OnLongClickListener。在 Activity 文件中使类继承状态变成：

```
public class ClickActivity extends AppCompatActivity implements
View.OnClickListener,View.OnLongClickListener
```

（3）实现长按事件处理接口中的方法，代码如下：

```
@Override
public boolean onLongClick(View v) {
    Toast.makeText(this, "您长按了一个 LinearLayout", Toast.LENGTH_LONG).show();
    return true;
}
```

这里也只是在长按之后显示一个 Toast。长按一个控件时会触发点击事件、触摸事件等，这里返回值的作用就在于此，当设置返回 true 时将不会发生连带触发的情况。

（4）在 intiView() 方法中获取 LinearLayout，并调用 setOnLongClickListener()方法进行注册：

```
linearLayout = (LinearLayout)findViewById(R.id.long_click_event);
linearLayout.setOnLongClickListener(this);
```

运行程序，发现 LinearLayout 确实是可以触发长按事件的，效果如图 4-9 所示。

图 4-9　触发长按事件

4.2.3 触摸事件

触摸事件是指触摸了某个控件而触发的事件，在 TextView、ImageView 等控件中比较常用，在 Button 中也会使用。下面将以 TextView 为实例来进行讲解。实例在点击事件的基础上进行如下修改即可：

（1）让 Activity 实现触摸事件的事件处理接口 View.OnTouchListener。

（2）实现触摸事件处理接口中的方法，代码如下：

```
@Override
public boolean onTouch(View v, MotionEvent event) {
    Toast.makeText(this, "您触摸了一个 TextView，它的坐标是：" +"X=" + event.getX() + ",Y=" +
event.getY(), Toast.LENGTH_LONG).show();
    return false;
}
```

这里也只是在触摸之后显示一个 Toast。触摸一个控件时必然会有一个按下与弹起的过程，这会连带触发点击事件，返回值的作用就在于此，当设置返回 true 时就不再发生连带触发的

情况。本例中使用默认的 false，以便于读者感知这种连带触发的效果。

另外，触摸事件与其他事件不同的地方在于它在用户触摸控件之后会返回一个 MotionEvent 对象，使用此对象可以获取控件的坐标。

（3）在 intiView()方法中已经获取 textView，直接调用 setOnTouchListener()方法进行注册即可：

```
textView.setOnTouchListener(this);
```

完成上述改动之后，运行程序，通过手指触摸 TextView 文本就可以触发触摸事件，如图 4-10 所示。但是在触摸事件被触发之后依旧会执行点击事件，如图 4-11 所示。



图 4-10　TextView 的触摸事件



图 4-11　在触摸事件被触发后还会执行点击事件

所以一般在开发中我们并不会在设置了点击事件后再去设置触摸事件。

4.2.4　按键事件

按键事件主要在 EditText 中使用，用于监听输入的内容。在点击事件基础上进行如下改动。

（1）在布局文件中加入一个 EditText，代码如下：

```
<EditText
    android:id="@+id/edti_event"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:textSize="26sp" />
```

（2）用 Activity 实现按键事件的事件处理接口 View.OnLongClickListener。

（3）实现按键事件处理接口中的方法，代码如下：

```
@Override
public boolean onKey(View v, int keyCode, KeyEvent event) {
    switch (event.getAction()) {
        case KeyEvent.ACTION_DOWN:
```

```
                Toast.makeText(this, "按键落下", Toast.LENGTH_LONG).show();
                break;
            case KeyEvent.ACTION_UP:
                EditText et = (EditText)v;
                Toast.makeText(this, "按键弹起，键入的是：" + et.getText().toString(),
Toast.LENGTH_LONG).show();
                break;
        }
        return false;
    }
```

实现的 onKey（View v, int keyCode, KeyEvent event）方法中有 3 个参数：View 参数在之前已经讲过，指代操作事件的 View 对象；keyCode 指的是输入按键的编码数字；event 用来表示按键的落下与弹起状态。在实例中，用 event.getAction()获取了按键的状态，并使用 switch 进行判断，对落下与弹起的状态进行处理。

这里的返回值很重要，如果返回的是 true，就意味着系统只处理我们代码中的这些事件，比如本例中的 Toast，而不再处理其他动作，如向 EditText 中写入文本。所以，一般情况下我们都会使用 false。读者在学习时一定要牢记这一点。

（4）在 initView 中获取 EditText 控件，并调用 setOnKeyListener()方法进行注册，代码如下：

```
editText = (EditText) findViewById(R.id.edti_event);
editText.setOnKeyListener(this);
```

运行程序，在向 EditText 中输入文本时捕获到的按键落下状态如图 4-12 所示。

当输入内容后，按键弹起，我们可以捕获到输入的内容，并通过 Toast 展示输入的内容，如图 4-13 所示。



图 4-12　按键事件中的按键落下状态



图 4-13　按键事件中的按键弹起状态

4.2.5　下拉列表的选中事件

处理下拉列表的选中事件自然只在下拉列表中使用，和上述几个事件的流程完全一致，

只是在下拉列表的开发过程上有所区别。由于在前面并没有讲解 Spinner 这个下拉列表控件，所以这里对 Spinner 的使用做一个详细的讲解。

（1）在布局文件中加入一个表示位置的下拉列表：

```
<Spinner
    android:layout_gravity="center"
    android:id="@+id/spinner_event"
    android:layout_width="200dp"
    android:layout_height="50dp"
    android:entries="@array/location" />
```

在 Spinner 的几个属性中，读者可能会对 android:entries 属性相对陌生一些。它是用来选择 Spinner 下拉选项内容的属性。在开发时，我们会在 res/values 中新建数组作为此属性的内容，向 Spinner 的下拉选项中注入数据。因此，我们在 res/values 中新建一个 location.xml，并建立一个数组：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string-array name="location">
        <item>虹口</item>
        <item>浦东</item>
        <item>闵行</item>
        <item>徐汇</item>
        <item>金山</item>
    </string-array>
</resources>
```

（2）使 Activity 实现下拉列表的事件处理接口 AdapterView.OnItemSelectedListener，其实是下拉列表某个条目的处理接口。然后实现该接口的处理方法，代码如下：

```
@Override
public void onItemSelected(AdapterView<?> parent, View view, int position, long id) {
    String value = parent.getItemAtPosition(position).toString();
    editText.setText("您的位置是："+value);
}

@Override
public void onNothingSelected(AdapterView<?> parent) {
}
```

特殊的是，它有两个实现方法，前者是选中某个条目之后的处理方法，后者是没有任何条目被选中的处理方法。当选中某条时，让它显示到 EditText 中。

（3）在 initView()方法中获取对象，并调用监听方法，代码如下：

```
spinner = (Spinner)findViewById(R.id.spinner_event);
spinner.setOnItemSelectedListener(this);
```

运行程序，选中其中一条，效果如图 4-14 所示。



图 4-14　下拉列表的选中事件

4.2.6　单选按钮的改变事件

单选按钮的改变事件自然只适用于单选按钮，所以首先要在布局文件中加入一个单选按钮，代码如下：

```
<RadioGroup
    android:id="@+id/sex_event"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:checkedButton="@+id/male">

    <RadioButton
        android:id="@+id/male"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="男" />

    <RadioButton
        android:id="@+id/female"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="女" />
</RadioGroup>
```

之后让 Activity 实现 RadioGroup.OnCheckedChangeListener 接口，并实现它的事件处理方法 onCheckedChanged()。实现代码如下：

```
@Override
public void onCheckedChanged(RadioGroup group, int checkedId) {
    RadioButton radioButton = (RadioButton)findViewById(checkedId);
    Toast.makeText(this, "您选择了："+radioButton.getText().toString(),
Toast.LENGTH_LONG).show();
}
```

这里实现的方法通过 checkId 这个被选中的单选按钮的 id 来获取单选按钮，并用 Toast 显示出选中的单选按钮的文本。

最后在 initView()方法中通过 findViewById()方法获取控件对象，并调用监听方法：

```
radioGroup = (RadioGroup) findViewById(R.id.sex_event);
radioGroup.setOnCheckedChangeListener(this);
```

当完成这些之后，运行程序，选中其中一个单选按钮，事件处理的效果如图 4-15 所示。

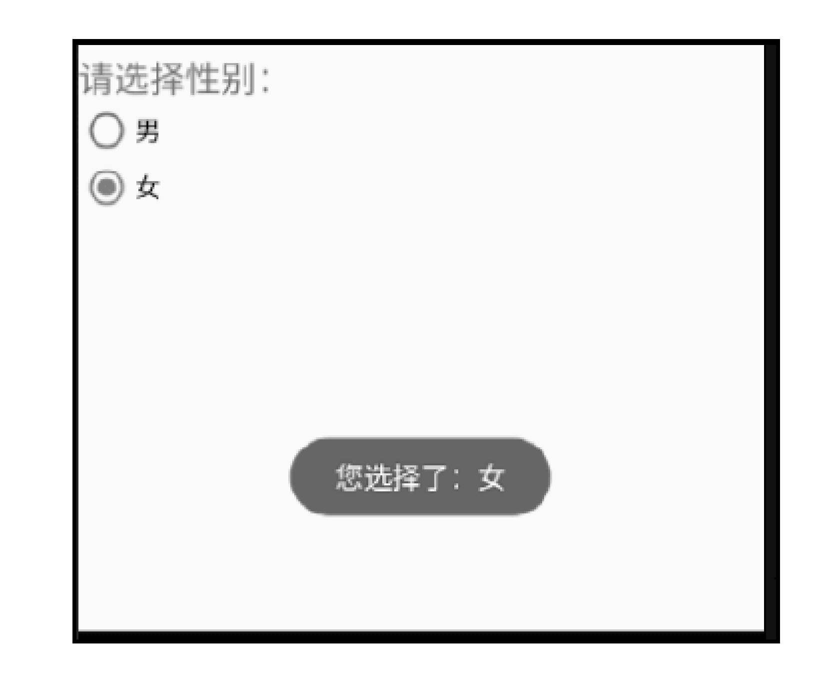


图 4-15　单选按钮的改变事件

4.2.7　焦点事件

焦点事件是指在多控件或多组件的状态下，操作某个控件就意味着该控件获取了焦点，同时也意味着之前获取焦点的控件失去了焦点。下面就用两个 EditText 来演示焦点事件。依旧使用上面实例的布局文件，此时只需让 Activity 实现焦点事件处理接口，同时实现对应的处理方法即可。

焦点事件处理接口是 View.OnFocusChangeListener，只需要 Activity 实现即可。下面实现它的事件处理方法 onFocusChange(View v, boolean hasFocus)，代码如下：

```
@Override
public void onFocusChange(View v, boolean hasFocus) {
    if (hasFocus) {
        Toast.makeText(this, "第一个 EditText 获得了焦点", Toast.LENGTH_LONG).show();
    } else {
        Toast.makeText(this, "第一个 EditText 失去了焦点", Toast.LENGTH_LONG).show();
    }
}
```

代码很简单，hasFocus 为 true 时，用 Toast 显示它获取了焦点，反之则显示它失去了焦点。

initView()方法中已经获取了该 EditText 的对象，所以只需调用监听方法就可以了：

```
editText.setOnFocusChangeListener(this);
```

运行程序，点击进入第二个 EditT 焦点时的效果如图 4-16 所示。重新进入第一个 EditText，效果如图 4-17 所示。



图 4-16　焦点事件—— 失去焦点



图 4-17　焦点事件—— 获得焦点

4.3 ListView 的使用

在 Android 开发中 ListView 是非常常用的组件，它以列表的形式展示具体内容，并且能够根据数据的长度自适应显示。它以垂直的方式排列其内部 item，其中的 item 可以被定义成各种复杂的界面，一般用于数据集的展示。ListView 在实际中使用的非常多，而且很多人在使用 ListView 时遇到的问题非常多，所以有必要单独用一节来讲解 ListView。

从上面的描述中可以总结出一个列表显示需要的三个要素：用来展示列表的 View，即 item；用来把数据映射到 ListView 的 item 上的适配器；具体的将被映射的字符串、图片、基本控件等数据。

适配器按照自定义程度分为 3 种：ArrayAdapter，SimpleAdapter 和通过继承 BaseAdapter 来自定义 Adapter。其中以 ArrayAdapter 最为简单，只能展示一行字。SimpleAdapter 有一定的扩充性，可以实现一定的自定义效果。自定义的 Adapter 则具有最好的扩展性，并能够实现各种自定义的效果。

下面一次讲解使用 3 种不同的适配器来实现的 ListView。

4.3.1 使用 ArrayAdapter 实现 ListView

使用 ArrayAdapter 来实现 ListView 的只能展示一行字，功能简单，实现也相对简单。首先在 Activity 对应的布局文件中加入 ListView 控件，就像加入 TextView 等其他普通控件一样。代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".activity.ListViewActivity">

    <ListView
        android:id="@+id/list"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />
</RelativeLayout>
```

本例中使用的是 ArrayAdapter，不需要去自建 item，因为系统给使用 ArrayAdapter 的 ListView 事先分配好了 item，只需要去调用就可以。下面是 Activity 中的代码：

```
package com.buaa.view.activity;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
```

```
import android.widget.ArrayAdapter;
import android.widget.ListView;
import com.buaa.view.R;
import java.util.List;

public class ListViewActivity extends AppCompatActivity {
    private ListView listView;
    String[] dataArr = new String[15];
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_list_view);
        initData();
        initView();
    }

    private void initView() {
        listView = (ListView) findViewById(R.id.list);
        ArrayAdapter<String> arrayAdapter = new ArrayAdapter<String>(this,
android.R.layout.simple_expandable_list_item_1, dataArr);
        listView.setAdapter(arrayAdapter);
    }

    private void initData() {
        for (int i = 0; i < 15; i++) {
            dataArr[i] = "ricky" + i;
        }
    }
}
```

在代码中，核心部分有两点。

第一，ArrayAdapter 适配器在初始化时需要传入 3 个参数，分别是对应的上下文对象，展示列表条目的 item，以及数据集。上下文对象传入的是当前对象，传入的 item 是系统自带的，需要使用 android.R.layout.simple_expandable_list_item_1 来获取。ArrayAdapter 初始化时需要的数据集必须是数组类型的，这里传入了一个字符串数组。

第二，在 Activity 中，通过使用 findViewById()方法获取了 ListView 类，ListView 通过调用 setAdapter()方法，设置了对应的适配器。

运行程序，一个使用 ArrayAdapter 实现的 ListView 就完成了，效果如图 4-18 所示。

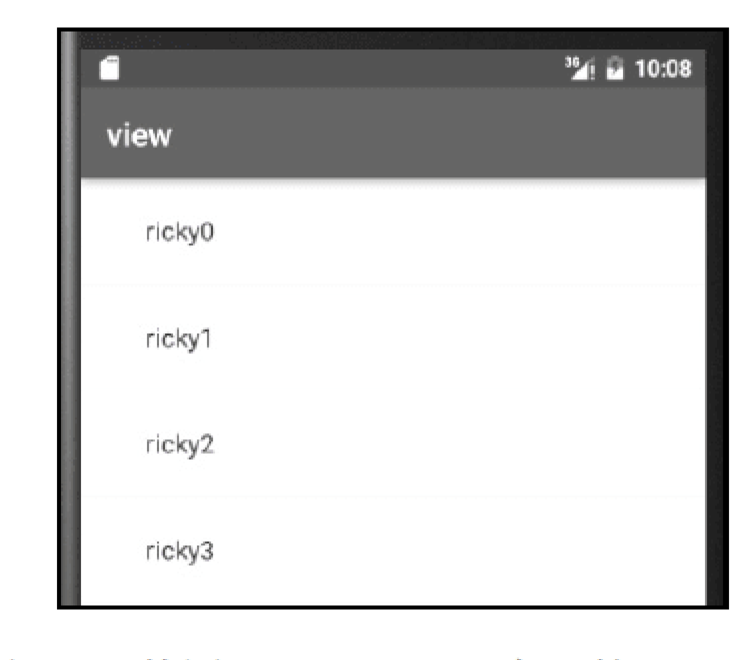


图 4-18 使用 ArrayAdapter 实现的 ListView

这种实现 ListView 的方式比较简单，但是能够应对的方式也简单。遇到更加复杂的状况只能使用其他两种方式。

4.3.2 使用 SimpleAdapter 实现 ListView

使用 SimpleAdapter 来实现 ListView 有一定的扩充性，可以实现一定的自定义效果，这种自定义的效果是通过创建 item 样式来实现的。创建一个名为 item_list.xml 的 item 布局文件，代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal">

    <ImageView
        android:id="@+id/item_draw"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_weight="1" />

    <TextView
        android:id="@+id/item_text"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:textSize="24sp" />
</LinearLayout>
```

这里展示了一个图像和一行文字。

将 ListView 加入 Activity 对应布局文件中的代码并不需要改变。而在 Activity 中，我们需要将 ArrayAdapter 修改为 SimpleAdapter，同时修改数据集。代码如下：

```
package com.buaa.view.activity;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import android.widget.SimpleAdapter;

import com.buaa.view.R;

import java.util.ArrayList;
```

```
import java.util.HashMap;
import java.util.List;
import java.util.Map;

public class ListViewActivity extends AppCompatActivity {

    private ListView listView;
    private List<Map<String, Object>> dataList = new ArrayList<Map<String, Object>>();
    private int[] itemIdArr = new int[]{R.id.item_text, R.id.item_draw};
    private String[] dataKeyArr = new String[]{"name", "draw"};

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_list_view);
        initData();
        initView();
    }

    private void initView() {
        listView = (ListView) findViewById(R.id.list);
        SimpleAdapter simpleAdapter = new SimpleAdapter(this,
                dataList,
                R.layout.item_list,
                dataKeyArr,
                itemIdArr);
        listView.setAdapter(simpleAdapter);
    }

    private void initData() {
        Map<String, Object> map;
        for (int i = 0; i < 15; i++) {
            map = new HashMap<String, Object>();
            map.put("name", "ricky" + i);
            map.put("draw", R.drawable.ic_launcher);
            dataList.add(map);
        }
    }
}
```

本实例代码和使用 ArrayAdapter 实现列表的代码大同小异，区别主要在于数据集的不同，以及实例化 Adapter 时的不同。SimpleAdapter 在实例化时需要传入 5 个参数，分别是上下文对象、List<Map<String, Object>>格式的数据集、item 的布局对象、数据集中 Map 键的数组、

item 中控件的 id 数组。当 SimpleAdapter 实例化完成后，系统会根据传入的两个数组自动将数据集注入 item 中。运行程序，效果如图 4-19 所示。

使用 SimpleAdapter 适配器实现的 ListView 能够实现更加复杂的状况，在实际开发中有时会使用到。但是，它有一个非常大的缺陷。想象一下，如果在每个 item 中都有一个按钮，需要给这个按钮添加点击事件，这就变得无法处理了。此时只能依靠使用 BaseAdapter 实现的 ListView。

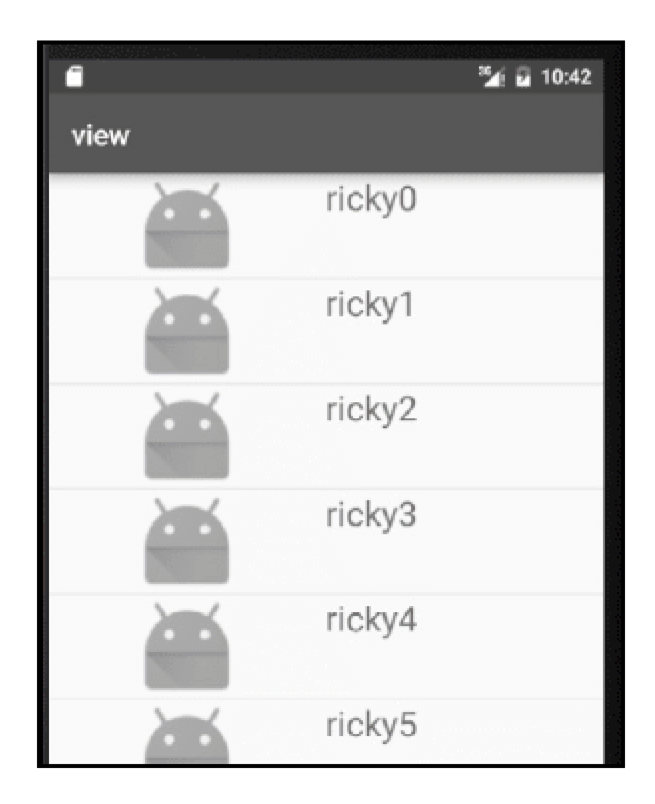


图 4-19　使用 SimpleAdapter 实现的 ListView

4.3.3　继承 BaseAdapter 自定义 Adapter 来实现 ListView

其实 SimpleAdapter 适配器已经能够满足很多情况了，但是由于它的一些缺陷，实际上在开发中使用最多、最广泛的还是通过继承 BaseAdapter 自定义 Adapter 的方式来实现 ListView。SimpleAdapter 和 ArrayAdapter 也是继承自 BaseAdapter 的，只是两种已经实现好的 BaseAdapter 而已。

和 SimpleAdapter 与 ArrayAdapter 一样，自定义 Adapter 需要至少实现 BaseAdapter 的 getCount()、getItem(int position)、getItemId(int position)以及 getView(final int position, View convertView, ViewGroup parent)这 4 个方法。下面通过实例说明这 4 个方法的作用。实例将在 res/layout 文件夹下创建一个 item_list.xml 布局文件，通过在这个 item 中加入一个 button 按钮来实现当点击该按钮时删除 item 的功能。item 的布局文件如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal">

    <ImageView
        android:id="@+id/item_draw"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_weight="1" />

    <TextView
        android:id="@+id/item_text"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:textSize="24sp" />
```

```
        <Button
            android:id="@+id/item_but"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="删除" />
</LinearLayout>
```

Activity 的布局文件依旧使用上例的布局文件。与上例不同的是，本实例中一个自定义的适配器类如下：

```
package com.buaa.view.adapter;

import android.content.Context;
import android.util.Log;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.BaseAdapter;
import android.widget.Button;
import android.widget.ImageView;
import android.widget.TextView;
import android.widget.Toast;

import com.buaa.view.R;

import java.util.List;
import java.util.Map;

public class MyAdapter extends BaseAdapter {
    private Context context;
    private List<Map<String, Object>> dataList;

    public MyAdapter(Context context, List<Map<String, Object>> dataList) {
        this.context = context;
        this.dataList = dataList;
    }

    @Override
    public int getCount() {
        return dataList.size();
    }

    @Override
```

```
        public Object getItem(int position) {
            return dataList.get(position);
        }

        @Override
        public long getItemId(int position) {
            return position;
        }

        @Override
        public View getView(final int position, View convertView, ViewGroup parent) {
            if (convertView == null) {
                convertView = LayoutInflater.from(context).inflate(R.layout.item_list, null);
            }

            ImageView img = (ImageView) convertView.findViewById(R.id.item_draw);
            TextView textView = (TextView) convertView.findViewById(R.id.item_text);
            Button del = (Button) convertView.findViewById(R.id.item_but);

            Map<String, Object> map = dataList.get(position);
            img.setImageResource((Integer) map.get("draw"));
            textView.setText((String) map.get("name"));

            del.setOnClickListener(new View.OnClickListener() {
                @Override
                public void onClick(View v) {
                    Toast.makeText(context, "删除了第" + position + "行", Toast.LENGTH_LONG).show();
                    dataList.remove(position);
                    notifyDataSetChanged();
                }
            });
            return convertView;
        }
    }
```

在这个自定义的适配器类中创建了一个用于接收数据集以及上下文对象的构造方法，并实现了 BaseAdapter 的 4 个方法。在这些方法中，getCount()、getItem(int position)、getItemId(int position)通过代码甚至方法名就能够很容易地理解它们的意义，这里就不做讲解了。

在一个完整的 ListView 第一次出现时，每个 item 都是空的，会调用 getView()方法去创建并返回一个 item（一个 View 对象）。这个过程就像代码中显示的，如果 convertView 是空的，就通过 LayoutInflater 的 inflate()方法获取这个 item 布局对象，并将其赋值给 convertView 对象。Inflate()方法与 findViewById()方法有些相似之处，不过一个是用来获取布局文件的对象，一个是用来获取控件的对象。当 convertView 对象被赋值之后，通过 convertView 对象调用

findViewById()方法来获取各个控件，并对控件进行一系列操作。此处对 Button 按钮添加了一个点击事件，当点击按钮时，将执行 dataList.remove(position)和 notifyDataSetChanged()两个方法。在 ListView 中，想要删除某条 item，需要执行的就是这两个方法，第一个方法是从数据集中将对应 item 的数据删除，第二个方法是用来观测数据集变化的，当数据集发生变化时，将重新执行 getView()方法，刷新界面，展示出删除该条 item 后的界面。

在以前的开发中，有人问过，如果我们有大量的数据需要显示，每个 item 都去 getView 中重复执行创建新的 view 的动作吗？答案是并不，系统会根据一个屏幕能够显示的 item 数量来执行创建 view 的动作。

当完成自定义 Adapter 的工作之后，在 Activity 中的使用就很简单了，只需要修改 initView()中的方法即可：

```
private void initView() {
    listView = (ListView) findViewById(R.id.list);
    MyAdapter myAdapter = new MyAdapter(this,
            dataList);
    listView.setAdapter(myAdapter);
}
```

运行应用，并尝试点击“删除”按钮，效果如图 4-20 所示。

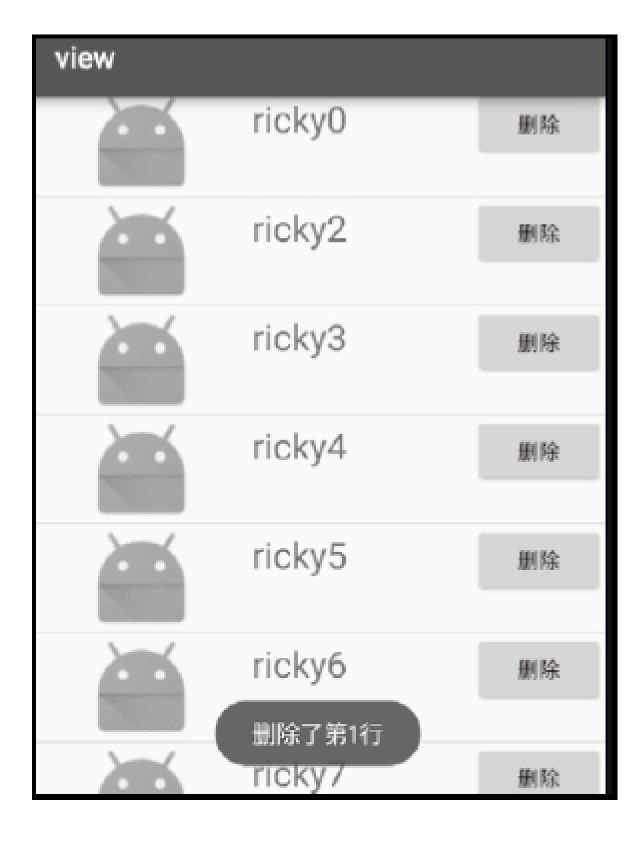


图 4-20　使用 BaseAdapter 实现的 ListView

通过实例发现这种自定义的适配器确实实现了点击“删除”按钮就能删除一条 item 的功能。这也说明了自定义的适配器要比前两种适配器强大得多。

继续分析实例，细心的读者会发现，按照上述代码以及我们之前描述的 getView()方法的原理会出现一种非常消耗内存的状况：每次删除一条记录或者上下滑动时会频繁地执行调用 findViewBy()方法去获取控件对象。所以有必要对上述代码进行改造。在开发过程发现最被开发者接受的一种方式是在适配器中添加一个静态内部类来保存 item 的控件对象，在 convertView 为空时，使用 convertView 的 setTag()方法来保存这个类对象。当 convertView 不为空时就直接使用 getTag()方法获取这个静态类的对象，然后依靠它获取 item 的控件对象。改动后的 getView()代码如下：

```
public View getView(final int position, View convertView, ViewGroup parent) {
    ViewHolder holder = null;
    if (convertView == null) {
        holder = new ViewHolder();
        convertView = LayoutInflater.from(context).inflate(R.layout.item_list, null);
        holder.textView = (TextView) convertView.findViewById(R.id.item_text);
        holder.button = (Button) convertView.findViewById(R.id.item_but);
        holder.imageView = (ImageView) convertView.findViewById(R.id.item_draw);
```

```
            convertView.setTag(holder);
        } else {
            holder = (ViewHolder) convertView.getTag();
        }

        Map<String, Object> map = dataList.get(position);
        holder.imageView.setImageResource((Integer) map.get("draw"));
        holder.textView.setText((String) map.get("name"));

        holder.button.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Toast.makeText(context, "删除了第" + position + "行", Toast.LENGTH_LONG).show();
                dataList.remove(position);
                notifyDataSetChanged();
            }
        });
        return convertView;
    }
```

静态类代码如下：

```
    static class ViewHolder {
        TextView textView;
        Button button;
        ImageView imageView;
    }
```

经过改动后，通过分析代码逻辑或者打印 log 的方式可以很明确地得知，确实不会重复去获取 item 类的控件对象了，而是重复使用已经保存好的 viewHolder 对象。上述改动后的适配器就是在开发实践中得出的最佳自定义适配器，希望读者能够掌握。

4.3.4 item 的事件处理

ListView 的 item 事件处理和其他控件的事件处理大同小异，都是在获取控件的对象之后调用事件监听，在监听方法中进行处理。下面用一个实例以点击事件和长按事件为例讲解 item 的事件处理。在实例中，让 Activity 实现 AdapterView.OnItemClickListener、AdapterView.OnItemLongClickListener 两个接口，并实行监听方法，代码如下：

```
    //点击事件
    @Override
    public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
        Toast.makeText(this, "删除第" + (position + 1) + "行", Toast.LENGTH_LONG).show();
        dataList.remove(position);
        myAdapter.notifyDataSetChanged();
```

```
    }

    //长按事件
    @Override
    public boolean onItemLongClick(AdapterView<?> parent, View view, int position, long id) {
        Toast.makeText(this, "您长按了第" + (position + 1) + "行", Toast.LENGTH_LONG).show();
        return true;
    }
```

这里设置点击事件为删除点击的 item，长按事件为弹出一个 Toast，显示长按的是第几行。在长按事件的监听方法中，默认的返回值为 true，此时长按事件结束时会触发点击事件，如果不想触发点击事件，只需要返回 false 即可。剩下的就是在 initView()方法中调用 listView 的 item 监听事件：

```
listView.setOnItemClickListener(this);
listView.setOnItemLongClickListener(this);
```

但是，此时运行程序，发现不管怎么点击或者长按，任何效果都没有触发。这是开发中很常见的一个问题，ListView 无法获取焦点。原因是在自定义的 item 中存在诸如 ImageButton、Button、CheckBox 等子控件（也可以说是 Button 或者 Checkable 的子类控件），此时这些子控件会将焦点获取到，而 item 是没有焦点的，所以 item 的点击事件没有响应。对于这个问题，很多书籍都没有提及，或者有的书籍提到此问题时给出了解决方案，例如让子控件失去焦点等，这都没有正确地解决问题。笔者结合实际工作中的一些经验，经过比较之后，在这里给出的解决办法是在 item 布局文件的根节点中加入 android:descendantFocusability="blocksDescendants"。android:descendantFocusability 属性的作用是当一个为 view 获取焦点时，定义 viewGroup 及其子控件之间的关系。它的值有三个：beforeDescendants，表示 viewgroup 会优先子类控件而获取到焦点；afterDescendants，表示 viewgroup 只有当其子类控件不需要获取焦点时才获取焦点；blocksDescendants 表示 viewgroup 会覆盖子类控件而直接获得焦点。

此时在此运行程序，点击事件和长按事件就可以执行了。点击事件的效果如图 4-21 所示，长按事件的效果如图 4-22 所示。

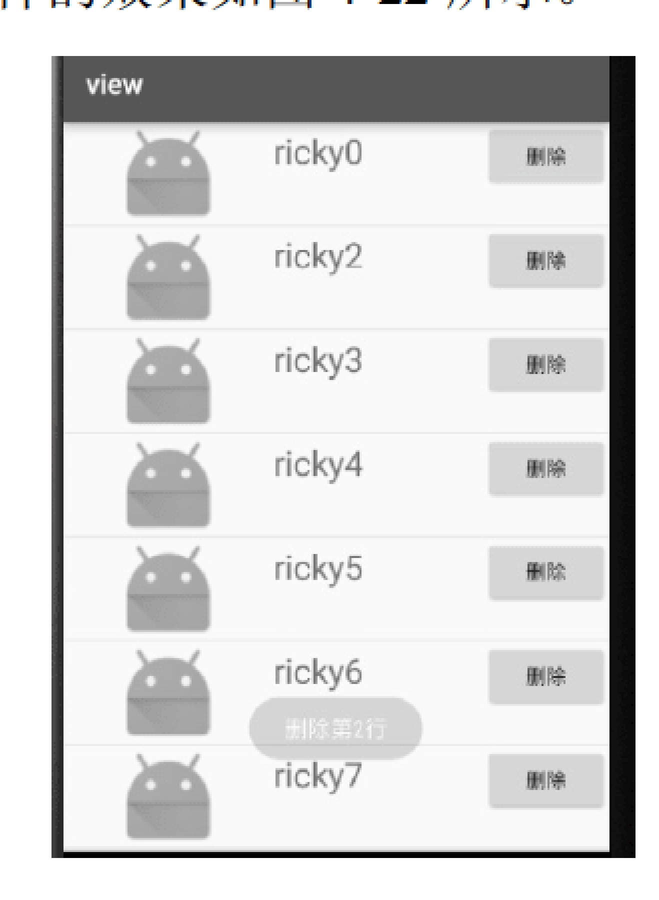


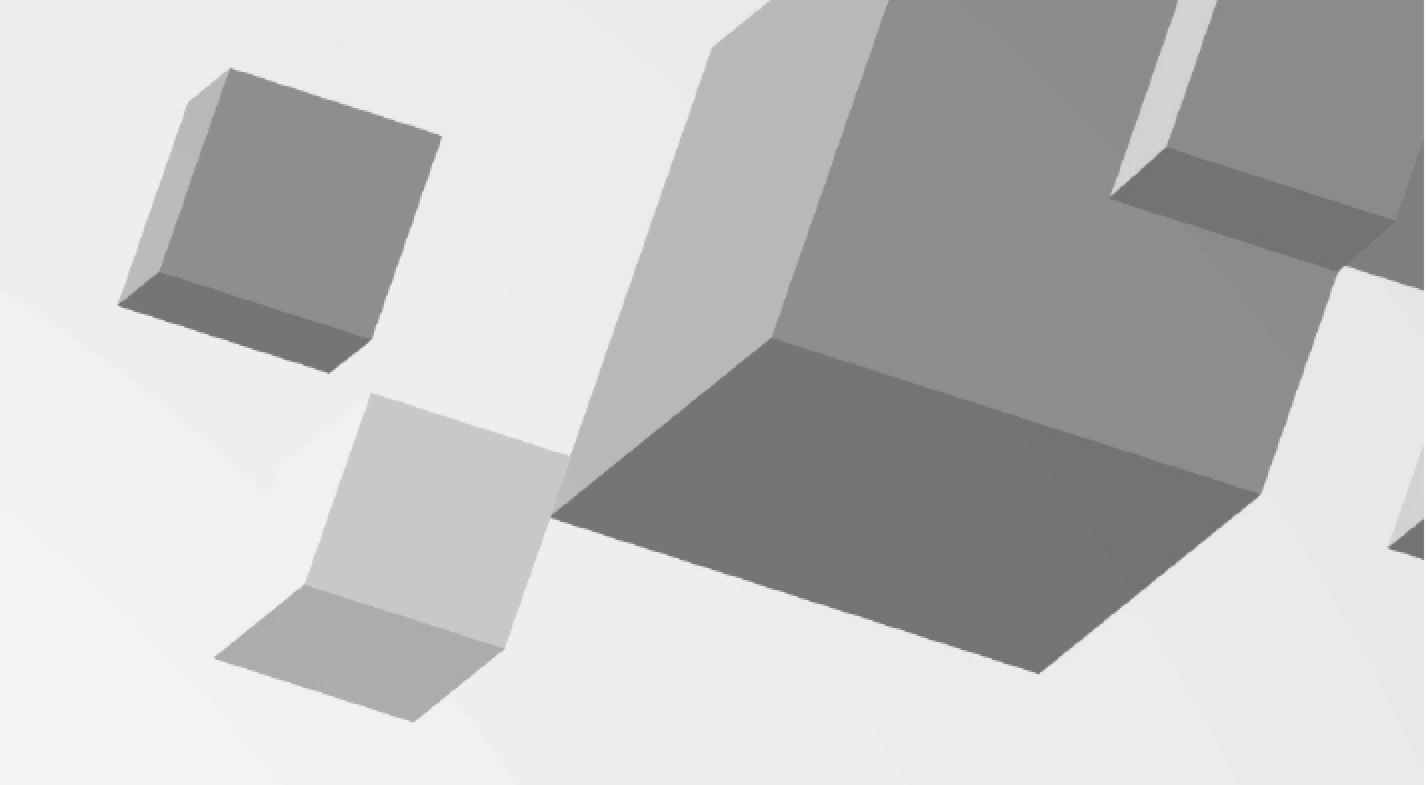
图 4-21　ListView 的 item 点击事件



图 4-22　ListView 的长按事件

4.4 小　　结

本章系统地讲解了在 Android 开发中常用的一些控件，并重点讲解了 ListView 这样一个在实际开发中经常使用的一个控件。同时，结合本节讲解的控件讲解了 Android 中的事件处理。关于事件处理，本章讲述的还不是非常深刻，主要关注于如何使用，希望读者在学习过程中多阅读 Android 文档，加强这方面的理解。另外，还讲解了在开发中困扰很多人的控件尺寸问题。



第 5 章 Fragment 详解

Android 系统是从 Android 3.0（API level 11）开始引入 Fragment 的。Fragment 可以作为 Activity 中的模块使用。这个模块有自己的布局，有自己的生命周期，单独处理自己的输入，在 Activity 运行的时候还可以加载或者移除 Fragment 模块。开发者也可以把 Fragment 设计成可以在多个 Activity 中复用的模块。

Fragment 还可以让 App 在原有性能基础上大幅度提高，并且使占用内存降低，同样的界面，Activity 占用内存比 Fragment 要多，Fragment 响应速度比 Activity 在中低端手机上快很多，甚至能达到好几倍！当开发的应用程序同时适用于平板电脑和手机时，可以利用 Fragment 实现灵活的布局，改善用户体验。

Fragment 是 Android 实际开发中又一个经常使用的知识，读者在学习本章时需要多做练习，熟加掌握。

5.1　Fragment 的创建与使用

学习一种技术最有效的方式是试着去使用它、理解它，本节将带着读者来创建与使用 Fragment。使用 Fragment 的方式有两种，一种是静态使用 Fragment，一种是动态使用 Fragment。

5.1.1　静态使用 Fragment

这是使用 Fragment 最简单的一种方式，把 Fragment 当成普通的控件，可以直接写在 Activity 的布局文件中。整个过程只需两步：一是继承 Fragment，重写 onCreateView 决定 Fragment 的布局；二是在 Activity 的布局文件中加入 Fragment，与普通的 View 一样。

下面直接用一个实例来展示如何使用。在这个实例中将使用两个 Fragment 作为 Activity 的布局，一个 Fragment 用于标题布局，一个 Fragment 用于内容布局。

MainActivity 代码：

```
package com.buaa.activity;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import com.buaa.fragment.R;

public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

MainActivity 的布局文件:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context="com.buaa.activity.MainActivity">

    <fragment
        android:id="@+id/id_fragment_title"
        android:name="com.buaa.fragment.TitleFragment"
        android:layout_width="fill_parent"
```

```
            android:layout_height="45dp" />

        <fragment
            android:id="@+id/id_fragment_content"
            android:name="com.buaa.fragment.ContentFragment"
            android:layout_width="fill_parent"
            android:layout_height="fill_parent" />
</LinearLayout>
```

在 ContentFragment 的代码里面加入一个 Button 的点击事件：

```
package com.buaa.fragment;

import android.os.Bundle;
import android.support.v4.app.Fragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.Button;
import android.widget.TextView;

public class ContentFragment extends Fragment {
    private Button contentBut;
    private TextView textView;

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                              Bundle savedInstanceState) {
        View fragmentView = inflater.inflate(R.layout.fragment_content, container, false);
        intiView(fragmentView);
        return fragmentView;
    }

    private void intiView(View fragmentView) {
        textView = (TextView) fragmentView.findViewById(R.id.contentText);
        contentBut = (Button) fragmentView.findViewById(R.id.contentBut);
        contentBut.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                textView.setText("中国羽毛球队获得了奥运会冠军");
            }
        });
    }
}
```

对应的布局文件如下：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context="com.buaa.fragment.ContentFragment">

    <TextView
        android:id="@+id/contentText"
        android:layout_width="match_parent"
        android:layout_height="100dp"
        android:text="@string/content"
        android:textSize="25sp" />

    <Button
        android:id="@+id/contentBut"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/addNews" />

</LinearLayout>
```

除此之外，还有一个 TitleFragment，代码如下：

```
package com.buaa.fragment;

import android.content.Context;
import android.net.Uri;
import android.os.Bundle;
import android.support.v4.app.Fragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

public class TitleFragment extends Fragment {
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        return inflater.inflate(R.layout.fragment_title, container, false);
    }

}
```

对应的布局文件如下：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#EE30A7"
    tools:context="com.buaa.fragment.TitleFragment">

    <TextView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:gravity="center"
        android:textSize="28sp"
        android:text="@string/news" />

</LinearLayout>
```

在上面的代码中可以很明显地看出 MainActivity 中只有加载布局文件的代码，因为处理 Button 按钮点击事件的代码在对应的 Fragment 中处理了。对所有的控件来说，事件处理都可以在对应的 Fragment 中进行，这样一来，代码的可读性、可维护性就大大提高了。在 Android 开发中经常会遇到这种可以分割成模块的界面，使用 Fragment 会变得非常方便。运行程序之后，效果如图 5-1 所示，点击“点击获取更多新闻”，就会通过操作 Fragment 来改变 TextView 的值，效果如图 5-2 所示。



图 5-1　使用 Fragment 进行布局的效果



图 5-2　对 Fragment 进行操作的效果

在上述代码中，不算上 Fragment 类，inflater.inflate(R.layout.fragment_content, container, false)这段代码可能很多读者也是第一次遇见，在这里解释一下。inflater 是 LayoutInflater 类的一个对象，它调用了 inflate 方法，这个方法类似于 findViewById，不同之处在于这个方法是用来找 res/layout/下的 xml 布局文件，并且实例化；而 findViewById()是找 xml 布局文件下的具体 widget 控件(如 Button、TextView 等)。

5.1.2　动态使用 Fragment

在上面的实例中应该已经能够看出使用 Fragment 的优势，但是静态使用 Fragment 的缺点也表露无遗。如果我们希望在多个 Fragment 间切换，就需要使用动态的方式去添加、更新以

及删除 Fragment。同样的，将以实例的形式展示如何动态地使用 Fragment。在本例中，除上例的两个 Fragment 之外，为实现动态切换，还新建了两个 Fragment，代码与上面的 Fragment 一致，就不再展示了。代码如下：

MainActivity 代码：

```
package com.buaa.activity;

import android.app.Activity;
import android.app.Fragment;
import android.app.FragmentManager;
import android.app.FragmentTransaction;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;

import com.buaa.fragment.ContentFragment;
import com.buaa.fragment.FirstPageFragment;
import com.buaa.fragment.PersonCenerFragment;
import com.buaa.fragment.R;

public class MainActivity extends Activity implements View.OnClickListener {
    private Button firstPage;
    private Button newsCenter;
    private Button personCenter;
    private Fragment contentFragment;
    private Fragment firstPageFragment;
    private Fragment personCenterFragment;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        initView();
    }

    private void initView() {
        firstPage = (Button) findViewById(R.id.firstPage);
        newsCenter = (Button) findViewById(R.id.newsCenter);
        personCenter = (Button) findViewById(R.id.personCenter);
        firstPage.setOnClickListener(this);
        newsCenter.setOnClickListener(this);
        personCenter.setOnClickListener(this);
        setDefaultFragment();
```

```
        }

        private void setDefaultFragment() {
            FragmentManager fm = getFragmentManager();
            FragmentTransaction transaction = fm.beginTransaction();
            firstPageFragment = new FirstPageFragment();
            transaction.replace(R.id.id_content, firstPageFragment);
            transaction.commit();
        }

        @Override
        public void onClick(View v) {
            FragmentManager fm = getFragmentManager();
            FragmentTransaction transaction = fm.beginTransaction();
            switch (v.getId()) {
                case R.id.firstPage:
                    if (firstPageFragment == null) {
                        firstPageFragment = new FirstPageFragment();
                    }
                    // 使用当前 Fragment 的布局替代 id_content 的控件
                    transaction.replace(R.id.id_content, firstPageFragment);
                    break;
                case R.id.newsCenter:
                    if (contentFragment == null) {
                        contentFragment = new ContentFragment();
                    }
                    transaction.replace(R.id.id_content, contentFragment);
                    break;
                case R.id.personCenter:
                    if (personCenterFragment == null) {
                        personCenterFragment = new PersonCenerFragment();
                    }
                    transaction.replace(R.id.id_content, personCenterFragment);
                    break;
            }
            transaction.commit();
        }
    }
```

对应的布局文件代码：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
```

```
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context="com.buaa.activity.MainActivity">

    <fragment
        android:id="@+id/id_fragment_title"
        android:name="com.buaa.fragment.TitleFragment"
        android:layout_width="match_parent"
        android:layout_height="45dp" />

    <FrameLayout
        android:id="@+id/id_content"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1"/>

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal">

        <Button
            android:id="@+id/firstPage"
            android:layout_width="0dp"
            android:layout_height="match_parent"
            android:layout_weight="1"
            android:text="微博首页" />

        <Button
            android:id="@+id/newsCenter"
            android:layout_width="0dp"
            android:layout_height="match_parent"
            android:layout_weight="1"
            android:text="新闻中心" />

        <Button
            android:id="@+id/personCenter"
            android:layout_width="0dp"
            android:layout_height="match_parent"
            android:layout_weight="1"
            android:text="个人中心" />
```

```
    </LinearLayout>
</LinearLayout>
```

运行程序，效果如图 5-3 所示，点击“新闻中心”按钮之后的效果如图 5-4 所示。



图 5-3　运行程序之后显示的效果



图 5-4　点击“新闻中心”按钮显示的效果

通过这个实例可以分析出，原来需要 3 个 Activity 的应用，现在只需要一个 Activity 加上 3 个 Fragment 就可以实现了，一方面节约硬件的消耗，另一方面使得代码更加简洁，耦合度大大降低了，即使需要再多的界面，也只需要创建一个 Fragment，在 Activity 中注册好这些 Fragment 就可以了，不再需要去修改原有的界面以及修改相关代码。

在实例代码中，出现了 FragmentManager 和 FragmentTransaction 两个新类，加上 Fragment 类，共 3 个类。读者以前没有接触这 3 个类，下面具体分析一下。相信通过下面的分析读者就能够明白上面的代码含义了。

5.1.3　使用 Fragment 时常用的类和方法

在使用 Fragment 时会经常使用 3 个类：android.app.Fragment 类，主要用于定义 Fragment；android.app.FragmentManager 类，主要用于在 Activity 中操作 Fragment；android.app.FragmentTransaction 类，保证一系列 Fragment 操作的原子性。在开发过程中，对于事件处理、其他相关的逻辑层处理很多都是在 Fragment 类中完成的。而在 Activity 中动态使用 Fragment 主要操作的都是 FragmentTransaction 方法。那么如何获取 FragmentTransaction 对象呢？一般来说通过如下两个步骤就可以了：

```
FragmentManage fragmentManage = getFragmentManager();
FragmentTransaction transaction = fm.benginTransatcion();
```

这样就获取了 FragmentTransaction 的对象，然后就可以操作 FragmentTransaction 的相关方法了。另外，读者需要注意，上述步骤中获取 FragmentManage 对象的方法在 v4 支持包中使用的是 getSupportFragmentManager()方法。表 5-1 展示的是 FragmentTransaction 类操作 Fragment 的相关方法。

表5-1　FragmentTransaction类中用来操纵Fragment的相关方法

add(int containerViewID,Fragment fragment)	向 Activity 中添加一个 Fragment
remove(int containerViewID,Fragment fragment)	从 Activity 中移除一个 Fragment，如果被移除的 Fragment 没有添加到回退栈，这个 Fragment 实例将会被销毁。
replace(int containerViewID,Fragment fragment)	使用另一个 Fragment 替换当前的，就是把 remove()和 add()合成一个方法
hide(int containerViewID,Fragment fragment)	隐藏当前的 Fragment，仅仅是设为不可见，并不会销毁
show(int containerViewID,Fragment fragment)	显示之前隐藏的 Fragment
detach(Fragment fragment)	将 View 从 UI 中移除,和 remove()不同,此时 fragment 的状态依然由 FragmentManager 维护
attach(Fragment fragment)	重建 View 视图，附加到 UI 上并显示
commit()	提交一个事务，这个方法必须最后使用

表 5-1 基本列出了所有操作 Fragment 的方式。一个事务从开启到提交可以进行多个添加、移除、替换等操作。这里的描述肯定是枯燥而不形象的，除了 replace()与 commit()方法在实例中使用过之外，其他都没有用过，读者一定要亲自操作一遍，须知实践出真知！

这里留下一个小问题给读者，如果在 FirstPageFragment 的布局文件中有一个 EditText，现在在这个 EditText 中已经有编辑的文本了，这时切换到 PersonCenterFragment，如果希望在下一次切换回 FirstPageFragment 时 EditText 中的数据依旧存在，该使用什么方法呢？

5.2　Fragment 生命周期

在讲解 Activity 时阐述了 Activity 的生命周期，现在讲解 Fragment 依旧要讲生命周期。只有理解生命周期，才能理解 Fragment 的生与死，理解它在不同状态下的处理方式。不理解生命周期肯定是写不出高质量程序的。

通过 5.1 节的内容，我们发现 Fragment 其实是绑定在 Activity 上的，那么它们的生命周期应该是有很大相关性的。事实也是这样，Fragment 与 Activity 相比，就是多了几个额外的生命周期回调方法：

- onAttach(Activity activity)方法，当 Fragment 与 Activity 发生关联时调用。
- onCreateView(LayoutInflater layoutInflater,ViewGroup vg,Bundle bundle)方法，创建该 Fragment 的视图时回调。
- onActivityCreated(Bundle bundle)方法，当 Activity 的 onCreate 方法返回时调用。
- onDestoryView()方法与 onCreateView 对应，当该 Fragment 的视图被移除时调用。
- onDetach()方法与 onAttach 相对应，当 Fragment 与 Activity 关联被取消时调用。

在此需要提醒读者，除了 onCreateView()方法外，如果重写了其他的所有方法，就必须调用父类对于该方法的实现。

图 5-5 是 Google 公司提供的 Fragment 的生命周期图和 Fragment 与 Activity 生命周期关联

图。通过此图也可以验证上文的论断，两者高度相关，Fragment 的生命周期只是比 Activity 的生命周期回调方法多几个。

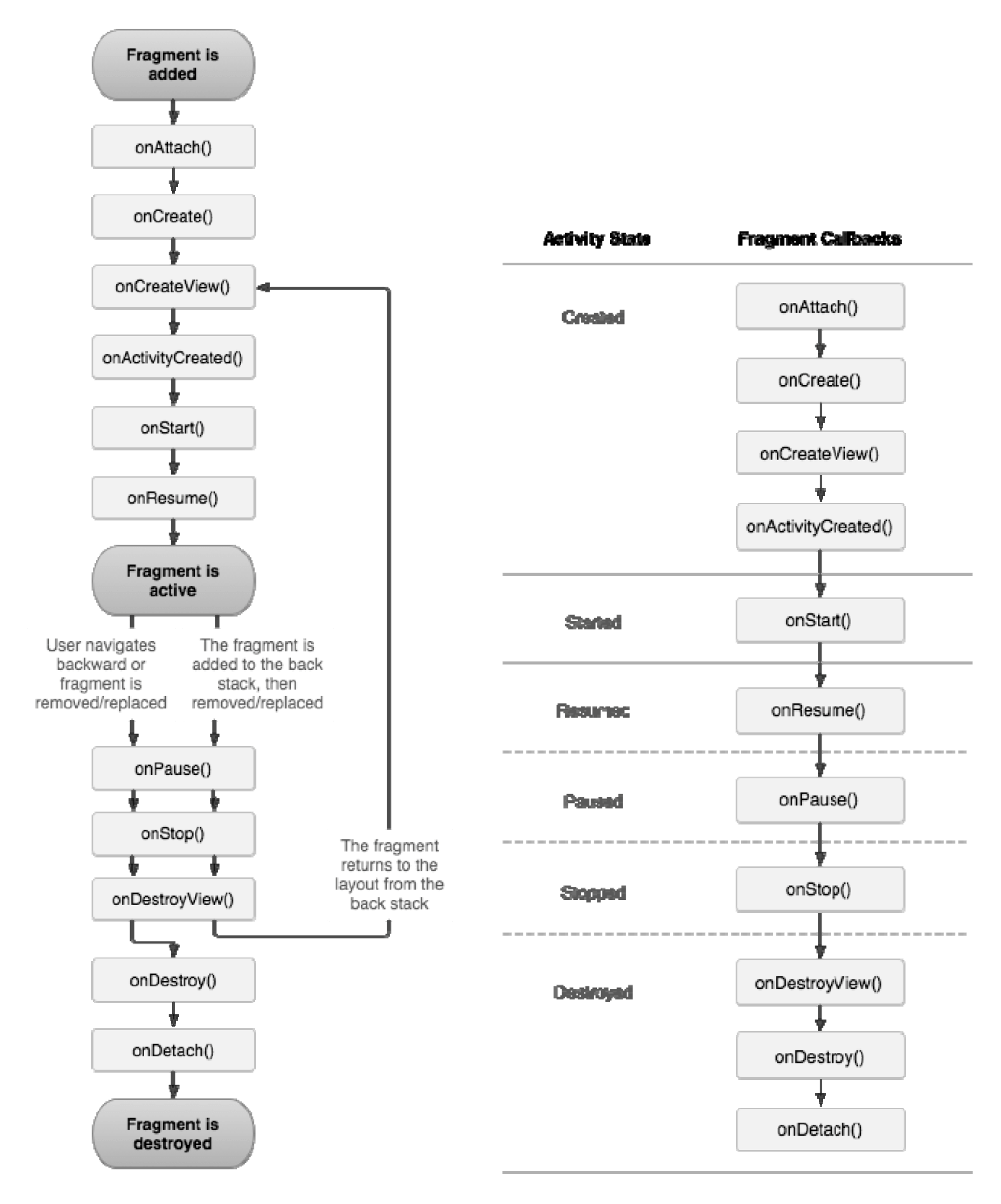


图 5-5　Fragment 的生命周期图和 Fragment 与 Activity 生命周期关联图

为了让读者能够更加直观地感受 Fragment 的生命周期，这里通过创建一个 Activity、一个 Fragment 并将 Fragment 与 Activity 绑定来演示生命周期状况。代码如下：

Activity 代码：

```
package com.buaa.activity;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.util.Log;

import com.buaa.fragment.LifeOfFragment;
import com.buaa.fragment.R;

public class FragmentLifeActivity extends AppCompatActivity {

    private LifeOfFragment lifeOfFragment;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_fragment_life);
        Log.e("FragmentLife", "Activity 的状态为 onCreate...");
        if (lifeOfFragment == null) {
            lifeOfFragment = new LifeOfFragment();
        }
        getSupportFragmentManager().beginTransaction().replace(R.id.life, lifeOfFragment).commit();
    }

    @Override
    protected void onStart() {
        // TODO Auto-generated method stub
        super.onStart();
        Log.e("FragmentLife", "Activity 的状态为 onStart...");
    }

    @Override
    protected void onResume() {
        // TODO Auto-generated method stub
        super.onResume();
        Log.e("FragmentLife", "Activity 的状态为 onResume...");
    }

    @Override
    protected void onStop() {
        // TODO Auto-generated method stub
```

```
        super.onStop();
        Log.e("FragmentLife", "Activity 的状态为 onStop...");
    }

    @Override
    protected void onPause() {
        // TODO Auto-generated method stub
        super.onPause();
        Log.e("FragmentLife", "Activity 的状态为 onPause...");
    }

    @Override
    protected void onDestroy() {
        // TODO Auto-generated method stub
        super.onDestroy();
        Log.e("FragmentLife", "Activity 的状态为 onDestroy...");
    }
}
```

Fragment 代码：

```
package com.buaa.fragment;

import android.os.Bundle;
import android.support.annotation.Nullable;
import android.support.v4.app.Fragment;
import android.util.Log;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

public class LifeOfFragment extends Fragment {
    @Override
    public void onStart() {
        Log.e("FragmentLife", "Fragment 的状态为 onStart...");
        super.onStart();
    }

    @Override
    public void onResume() {
        Log.e("FragmentLife", "Fragment 的状态为 onResume...");
        super.onResume();
    }
```

```
@Override
public void onPause() {
    Log.e("FragmentLife", "Fragment 的状态为 onPause...");
    super.onPause();
}

@Override
public void onStop() {
    Log.e("FragmentLife", "Fragment 的状态为 onStop...");
    super.onStop();
}

@Override
public void onDestroy() {
    Log.e("FragmentLife", "Fragment 的状态为 onDestroy...");
    super.onDestroy();
}

@Override
public void onDetach() {
    Log.e("FragmentLife", "Fragment 的状态为 onDetach...");
    super.onDetach();
}

@Override
public void onSaveInstanceState(Bundle outState) {
    Log.e("FragmentLife", "Fragment 的状态为 onSaveInstanceState...");
    super.onSaveInstanceState(outState);
}

@Override
public void onActivityCreated(Bundle savedInstanceState) {
    Log.e("FragmentLife", "Fragment 的状态为 onFragmentCreated...");
    super.onActivityCreated(savedInstanceState);
}

@Override
public void onDestroyView() {
    Log.e("FragmentLife", "Fragment 的状态为 onDestroyView...");
    super.onDestroyView();
}

@Override
```

```
        public void onCreate(Bundle savedInstanceState) {
            Log.e("FragmentLife", "Fragment 的状态为 onCreate...");
            super.onCreate(savedInstanceState);
        }

        @Nullable
        @Override
        public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {
            Log.e("FragmentLife", "Fragment 的状态为 onCreateView...");
            return inflater.inflate(R.layout.fragment_life_of, container, false);
        }
    }
```

通过运行程序，我们就可以清晰地发现不同状况下 Fragment 的生命周期以及 Fragment 生命周期与 Activity 生命周期之间的关系了。经过打开应用、横竖屏切换、退出应用、进入其他 Activity、重新回到 Activity 等多种操作之后，观察打印的 log，经过总结，发现 Activity 生命周期与 Fragment 的生命周期确实是高度相关的。Log 如下：

onCreate()过程：

```
31812-31812/com.buaa.fragment E/FragmentLife: Activity 的状态为 onCreate...
31812-31812/com.buaa.fragment E/FragmentLife: Fragment 的状态为 onCreate...
31812-31812/com.buaa.fragment E/FragmentLife: Fragment 的状态为 onCreateView...
31812-31812/com.buaa.fragment E/FragmentLife: Fragment 的状态为 onActivityCreated...
```

onStart()的过程：

```
19004-19004/com.buaa.fragment E/FragmentLife: Fragment 的状态为 onStart...
19004-19004/com.buaa.fragment E/FragmentLife: Activity 的状态为 onStart...
```

onResume()的过程：

```
19004-19004/com.buaa.fragment E/FragmentLife: Activity 的状态为 onResume...
19004-19004/com.buaa.fragment E/FragmentLife: Fragment 的状态为 onResume...
```

onPause()的过程：

```
19004-19004/com.buaa.fragment E/FragmentLife: Fragment 的状态为 onPause...
19004-19004/com.buaa.fragment E/FragmentLife: Activity 的状态为 onPause...
```

onStop()的过程：

```
19004-19004/com.buaa.fragment E/FragmentLife: Fragment 的状态为 onStop...
19004-19004/com.buaa.fragment E/FragmentLife: Activity 的状态为 onStop...
```

onDestroy()的过程：

```
19004-19004/com.buaa.fragment E/FragmentLife: Fragment 的状态为 onDestroyView...
19004-19004/com.buaa.fragment E/FragmentLife: Fragment 的状态为 onDestroy...
19004-19004/com.buaa.fragment E/FragmentLife: Fragment 的状态为 onDetach...
19004-19004/com.buaa.fragment E/FragmentLife: Activity 的状态为 onDestroy...
```

正如上文所说，除个别状态下的生命周期不一致之外，Fragment 与 Activity 的生命周期是相当一致的。

5.3 ListFragment 的使用

在开发过程中，经常会使用 ListView 控件，也就会经常出现在 Fragment 中使用 ListView 的状况。Android 在支持 Fragment 的时候就提供了 ListFragment，以实现在 Fragment 使用 ListView 的需求。使用 ListFragment，只能使用 SimpleAdapter 或者 SimpleCursorAdapter 作为适配器。本节就讲解如何使用 ListFragment。

使用 Android Studio 创建一个 Fragment，并使它继承 ListFragment。在创建出 Fragment 文件的同时，Studio 会同时创建一个布局文件。修改 Fragment 文件并在布局文件中添加 ListView 控件，同时创建一个 ListView 的 item 文件，代码如下：

```
package com.buaa.fragment;

import android.os.Bundle;
import android.support.v4.app.ListFragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ListView;
import android.widget.SimpleAdapter;
import android.widget.Toast;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;


public class NewsListFragment extends ListFragment {

    private ListView listView;
    private SimpleAdapter adapter;

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                              Bundle savedInstanceState) {
        View view = inflater.inflate(R.layout.fragment_news_list, container, false);
        listView = (ListView) view.findViewById(android.R.id.list);
        return view;
    }
```

```
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        String[] strings = {"乌克兰危机持续发酵",
                            "南海仲裁中国宣布不参与不承认",
                            "蔡英文当选百日毫无政绩",
                            "北京航空航天大学不再设围墙",
                            "清华大学出版社发行 Android 书籍"};
        adapter = new SimpleAdapter(getActivity(), getData(strings),
                R.layout.item_list, new String[]{"title"}, new int[]{R.id.news_title});
        setListAdapter(adapter);
    }

    @Override
    public void onListItemClick(ListView lv, View v, int position, long id) {
        super.onListItemClick(lv, v, position, id);
        Toast.makeText(getActivity(), "欢迎阅读本条新闻", Toast.LENGTH_LONG).show();
    }

    private List<? extends Map<String, ?>> getData(String[] strs) {
        List<Map<String, Object>> list = new ArrayList<>();
        for (int i = 0; i < strs.length; i++) {
            Map<String, Object> map = new HashMap<>();
            map.put("title", strs[i]);
            list.add(map);
        }
        return list;
    }
}
```

布局文件如下：

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.buaa.fragment.NewsListFragment">

    <ListView
        android:id="@id/android:list"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"></ListView>

</FrameLayout>
```

Item 文件：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        android:id="@+id/news_title"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="25sp" />
</LinearLayout>
```

这里需要提醒读者，在 Fragment 布局文件中 ListView 控件的 id 必须为 android:id="@id/android:list"。同时在 Fragment 类文件中，获取此 ListView 使用的是 android.R.id.list，而不是 R.id.list 或者 R.id.android:list。代码的其他部分在上一章都已有讲解，就不再重复讲解了。之后秩序创建 Activity 类，将 ListFragment 类与之绑定即可，这部分内容与前文并无区别，不再详述。

运行程序，效果如图 5-6 所示。

与第 4 章使用 ListView 和 Activity 实现的效果是一致的。只是此时使用的是 ListFragment 来实现的。经过对比会发现，使用 ListFragment 进行开发会更加简洁，同时使用 Android Studio 工具中的内存分析工具可以发现当需要多个 ListView 时 ListFragment 方式更节约内存。



图 5-6　ListFragment 的显示效果

5.4　用 DialogFragment 创建对话框

DialogFragment 是一种特殊的 Fragment，用于在 Activity 的内容之上展示一个模态的对话框。典型的应用有：展示警告框、输入框、确认框等。使用 DialogFragment 来管理对话框，当旋转屏幕和按下后退键时可以更好地管理其生命周期，它和 Fragment 有着基本一致的生命周期。DialogFragment 允许开发者把 Dialog 作为内嵌的组件进行重用。

使用 DialogFragment 至少需要实现 onCreateView 或者 onCreateDIalog 方法。onCreateView 利用定义的 xml 布局文件展示 Dialog，onCreateDialog 利用 AlertDialog 或者 Dialog 创建出来。

下面通过实例说明如何通过继承 DialogFragment 类，重写 onCreateView 或者 onCreateDialog 方法来实现对话框。

5.4.1 通过重写 onCreateView 方法来实现对话框

要实现对话框，首先需要实现一个对话框的布局文件，这里给出代码实例：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="300dp"
    android:layout_height="wrap_content"
    android:gravity="center"
    android:orientation="vertical"
    tools:context="com.buaa.fragment.LoginFragment">

    <EditText
        android:layout_width="120dp"
        android:layout_height="wrap_content"
        android:inputType="text" />

    <EditText
        android:layout_width="120dp"
        android:layout_height="wrap_content"
        android:inputType="textPassword" />

</LinearLayout>
```

然后创建一个继承自 DialogFragment 类的 Fragment，并且实现 onCreateView 方法：

```
public class LoginFragment extends DialogFragment {
    private EditText name;
    private EditText password;
    private Button login;

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                              Bundle savedInstanceState) {
        View view = inflater.inflate(R.layout.fragment_login, container, false);
        initView(view);
        return view;
    }

    private void initView(View view) {
        name = (EditText) view.findViewById(R.id.name);
        password = (EditText) view.findViewById(R.id.password);
        login = (Button) view.findViewById(R.id.login);
```

```
            login.setOnClickListener(new View.OnClickListener() {
                @Override
                public void onClick(View v) {
                    Toast.makeText(getActivity(), "用户名为："
                            +name.getText().toString() + ",密码为：" +
                            password.getText().toString(), Toast.LENGTH_LONG).show();
                }
            });
        }
    }
```

在 MainActivity 中绑定 Fragment，并展示 Dialog：

```
public class MainActivity extends Activity implements View.OnClickListener {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        showDialog();
    }

    public void showDialog() {
        LoginFragment loginFragment = new LoginFragment();
        loginFragment.show(getFragmentManager(), "登录");
    }
}
```

此处代码并无艰深难懂之处，与前几节内容也都大同小异，无非是继承的 Fragment 类不同而已。运行程序的效果如图 5-7 所示。



图 5-7　DialogFragment 通过重写 onCreateView 方法来实现对话框

通过上述的两个步骤即可创建一个 Dialog，读者可以对照上述实例多加练习。

5.4.2 通过重写 onCreateDialog 方法来实现对话框

重写 onCreateDialog 方法来实现对话框与通过重写 onCreateView 方法的区别只在于 Fragment 内的这个重写方法，Activity 类和布局文件都不需要变化。使用重写 onCreateDialog 方法的 DialogFragment 类的代码如下：

```
public class LoginFragment extends DialogFragment {
    private EditText name;
    private EditText password;

    @Override
    public Dialog onCreateDialog(Bundle savedInstanceState) {
        LayoutInflater inflater = getActivity().getLayoutInflater();
        View view = inflater.inflate(R.layout.fragment_login, null);
        name = (EditText) view.findViewById(R.id.name);
        password = (EditText) view.findViewById(R.id.password);
        AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());
        builder.setView(view)
                .setPositiveButton("登录",
                        new DialogInterface.OnClickListener() {
                            @Override
                            public void onClick(DialogInterface dialog, int id) {
                                Toast.makeText(getActivity(), "用户名为："
                                        + name.getText().toString() + ",密码为：" +
                                        password.getText().toString(),
Toast.LENGTH_LONG).show();
                            }
                        }).setNegativeButton("取消", null);
        return builder.create();
    }
}
```

通过运行程序发现，此时的效果和重写 onCreateView 方法的效果不太一样。当点击“登录”按钮之后，输入框会消失，同时软键盘也会消失。效果如图 5-8 所示。



图 5-8 DialogFragment 通过重写 onCreateDIalog 方法实现对话框

对于上述两种通过 DialogFragment 实现对话框的方法，可能部分读者会认为这并没有什么，甚至会说变得更复杂了。它与原生的 Dialog 相比，有什么优势呢？通过旋转屏幕，我们会发现对话框依旧在，数据不见了，这与原生的 Dialog 并无区别。事实上，只要在编辑框内输入内容，就会被保存下来，当然在上述的两个实例中并没有被保存，其中原因等下一节再解释。任意旋转屏幕时，对话的用户数据依旧被保存，用户的体验提升就好了很多。

5.5　Fragment 在开发中遇到的一些常见问题

Fragment 在开发中经常使用，自然在开发中也会遇到很多问题。本节将就旋转屏幕问题、Fragment 返回栈、Fragment 与 Activity 数据通信 3 个最常见问题进行讲解。

5.5.1　旋转屏幕问题

这里所说的屏幕旋转问题的实质是运行时配置发生变化，最常见的就是屏幕旋转。

除了 5.4 节结尾时所说的旋转屏幕时 FragmentDialog 的数据没有被保存之外，在 5.1 节的第二个实例中也存在这样的问题。当我们点击进入“新闻中心”时，页面显示正常，但是这个时候旋转屏幕，会惊奇地发现页面变成了“微博首页”。

思考一下 Fragment 的生命周期，当旋转屏幕时，其实 Activity 经历了一次销毁重建的过程，自然 Fragment 也要经历这样一个过程，FragmentDialog 中的数据必然要消失了，因为旋转屏幕之后我们看到的 FragmentDialog 已经不是原来的 FragmentDialog 了。5.1 节中的实例也是同样的道理，当旋转屏幕后，Activity 实例被销毁并重新调用 onCreate()方法，这时当然会调用默认的 Fragment 覆盖在之前的 Fragment。

说到这里，读者应该明白其中的原因了。知道原因后，解决它就相当简单了。只需要在设置默认 Fragment 时判断 Bundle 的对象是否为空即可。改变实例中的部分代码：

```
private void setDefaultFragment(Bundle onSaveInstanceState) {
    if (onSaveInstanceState == null) {
        fm = getFragmentManager();
        transaction = fm.beginTransaction();
        firstPageFragment = new FirstPageFragment();
        transaction.replace(R.id.id_content, firstPageFragment);
        transaction.commit();
    }
}
```

解决这个问题的方法很简单，但是如果读者不理解 Fragment 与 Activity 的生命周期，肯定不会想到要这么解决。5.4 节结尾的问题在这里已经解决了，具体的操作请读者自行完成。

5.5.2　Fragment 返回栈

还是以 5.1 节的第二个实例为例，当运行程序之后，点击不同的按钮，进入不同的 Fragment

界面。这时如果想要回到上一个 Fragment 应该怎么办呢？试着尝试按 back 键，发现程序直接退出了（只有一个 Activity 的情况）。

原因在于这些 Fragment 并没有加入返回栈。在多个 Activity 时，按 back 键会返回上一个 Activity，这是因为有一个 Activity 的栈在管理这些 Activity，但是 Fragment 并不是。如果想要实现同 Activity 一样的效果，就必须把 Fragment 加入返回栈中。修改该实例中的代码：

```
@Override
public void onClick(View v) {
    fm = getFragmentManager();
    transaction = fm.beginTransaction();
    switch (v.getId()) {
        case R.id.firstPage:
            if (firstPageFragment == null) {
                firstPageFragment = new FirstPageFragment();
            }
            // 使用当前 Fragment 的布局替代 id_content 的控件
            transaction.replace(R.id.id_content, firstPageFragment);
            break;
        case R.id.newsCenter:
            if (contentFragment == null) {
                contentFragment = new ContentFragment();
            }
            transaction.replace(R.id.id_content, contentFragment);
            break;
        case R.id.personCenter:
            if (personCenterFragment == null) {
                personCenterFragment = new PersonCenerFragment();
            }
            transaction.replace(R.id.id_content, personCenterFragment);
            break;
    }
    transaction.addToBackStack(null);
    transaction.commit();
}
```

这里只是加入了 transaction.addToBackStack(null);这样一行代码，运行程序之后，发现确实可以解决上面所说的问题。但是，这时如果在某一个 Fragment 页面中有一个编辑框，用户正在编辑或者从 Fragment 中向编辑框中注入了数据（如图 5-9 所示）。点击进入其他的 Fragment，然后按 back 键，返回到当前 Fragment。会有什么效果，数据会保存吗？很显然，并没有，如图 5-10 所示。



图 5-9　从 Fragment 中向编辑框中注入数据



图 5-10　重入 Fragment 时的编辑框内数据丢失了

这是因为在代码中使用的是 replace()方法，在前文中已经讲过，此方法实际上是 add()方法和 remove()方法的合并。调用 remove()方法会将需要 remove 的 Fragment 从 Activity 中移除，如果被移除的 Fragment 没有添加到回退栈，那么这个 Fragment 实例将会被销毁。即使此 Fragment 加入了返回栈，Fragment 实例没有被销毁，对应的视图也会被销毁。因此要想解决这个问题，需要使用 add()方法、show()方法、hide()方法、attach()方法来进行操作。同时，还需要拿到 Fragment 列表而使用的 FragmentManager 类的 getFragments()，只有在 v4 支持包以及更高版本支持包中含有，因此此处使用 v4 支持包的 Fragment 类。其他各类代码基本不变，只需将 Fragment 类的包换掉，Activity 类中改动较大的代码如下：

```
package com.buaa.activity;

import android.os.Bundle;
import android.support.v4.app.Fragment;
import android.support.v4.app.FragmentManager;
import android.support.v4.app.FragmentTransaction;
import android.support.v7.app.AppCompatActivity;
import android.util.Log;
import android.view.View;
import android.widget.Button;

import com.buaa.fragment.ContentFragment;
import com.buaa.fragment.FirstPageFragment;
import com.buaa.fragment.PersonCenerFragment;
import com.buaa.fragment.R;

import java.util.List;

public class MainActivity extends AppCompatActivity implements View.OnClickListener {
    private Button firstPage;
    private Button newsCenter;
    private Button personCenter;
    private Fragment contentFragment;
    private Fragment firstPageFragment;
    private Fragment personCenterFragment;
    private FragmentManager fm;
    private FragmentTransaction transaction;
```

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    initView(savedInstanceState);
}

private void initView(Bundle savedInstanceState) {
    firstPage = (Button) findViewById(R.id.firstPage);
    newsCenter = (Button) findViewById(R.id.newsCenter);
    personCenter = (Button) findViewById(R.id.personCenter);
    firstPage.setOnClickListener(this);
    newsCenter.setOnClickListener(this);
    personCenter.setOnClickListener(this);
    setDefaultFragment(savedInstanceState);
}

private void setDefaultFragment(Bundle onSaveInstanceState) {
    if (onSaveInstanceState == null) {
        fm = getSupportFragmentManager();
        transaction = fm.beginTransaction();
        firstPageFragment = new FirstPageFragment();
        transaction.add(R.id.id_content, firstPageFragment);
        transaction.commit();
    }
}

@Override
public void onClick(View v) {
    fm = getSupportFragmentManager();
    transaction = fm.beginTransaction();
    switch (v.getId()) {
        case R.id.firstPage:
            if (firstPageFragment == null) {
                backFragment(null);
                firstPageFragment = new FirstPageFragment();
                transaction.add(R.id.id_content, firstPageFragment);
            } else {
                backFragment(firstPageFragment);
            }
            break;
        case R.id.newsCenter:
```

```
                if (contentFragment == null) {
                    backFragment(null);
                    contentFragment = new ContentFragment();
                    transaction.add(R.id.id_content, contentFragment);
                } else {
                    backFragment(contentFragment);
                }
                break;
            case R.id.personCenter:
                if (personCenterFragment == null) {
                    backFragment(null);
                    personCenterFragment = new PersonCenerFragment();
                    transaction.add(R.id.id_content, personCenterFragment);
                } else {
                    backFragment(personCenterFragment);
                }
                break;
        }
        transaction.addToBackStack(null);
        transaction.commit();
    }

    private void backFragment(Fragment fragment) {
        List<Fragment> fragmentList = fm.getFragments();
        for (Fragment frag : fragmentList) {
            if (frag != null && !frag.isHidden() && frag != fragment) {
                transaction.hide(frag);
            }
        }
        if (fragment != null) {
            if (fragment.isHidden()) {
                transaction.show(fragment);
            }
            if (fragment.isDetached()) {
                transaction.attach(fragment);
            }
        }
    }
}
```

运行程序，发现前面所说的数据消失的状况再也没有出现。在上面的代码中重点是 backFragment(Fragment fragment)方法，在这里通过 fm.getFragments()获取了当前 Fragment 管理器中的 Fragment 列表，然后通过 frag != null && !frag.isHidden() && frag != fragment 判断出

列表中的 Fragment 是不是点击要进入的 Fragment 或者没有被隐藏的 Fragment。如果既没有被隐藏也不是想要进入的 Fragment，就将其隐藏。再通过 fragment.isDetached()判断想要进入的 Fragment 有没有绑定好 View 界面，如果没有就将它们绑定。fragment.isDetached()这一步很重要，如果没有这一步，将会出现白屏的状况。

5.5.3 Fragment 与 Activity 之间的数据通信

在开发时，经常需要将 Fragment 中的数据传递给 Activity，方法很多，比如 Activity 中包含自己管理的 Fragment 的引用，可以通过引用直接访问所有 Fragment 的 public 方法；如果 Activity 中未保存任何 Fragment 的引用，那么没关系，每个 Fragment 都有一个唯一的 TAG 或者 ID,可以通过 getFragmentManager.findFragmentByTag()或者 findFragmentById()获得任何 Fragment 实例，然后进行操作；在 Fragment 中可以通过 getActivity 得到当前绑定的 Activity 的实例，然后进行操作。在开发中最常用的方法如下：

```
package com.buaa.fragment;

import android.os.Bundle;
import android.support.v4.app.Fragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.Button;
import android.widget.EditText;

public class DataFragment extends Fragment {

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                                Bundle savedInstanceState) {
        View view = inflater.inflate(R.layout.fragment_data, container, false);
        initView(view);
        return view;
    }

    private void initView(View view) {
        final EditText password = (EditText) view.findViewById(R.id.data);
        Button login = (Button) view.findViewById(R.id.login_data);

        login.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                if (getActivity() instanceof DataFragmentCallBack) {
                    ((DataFragmentCallBack) getActivity()).
```

```
                        onLoginClick(password.getText().toString());
                }
            }
        });
    }

    public interface DataFragmentCallBack {
        void onLoginClick(String data);
    }
}
```

在 Fragment 中定义一个接口，在接口中定义一个处理点击事件的方法，我们将通过这个方法向 Activity 中传递数据。Activity 代码如下：

```
package com.buaa.fragment;

import android.os.Bundle;
import android.support.v4.app.Fragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.Button;
import android.widget.EditText;

public class DataFragment extends Fragment {

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        View view = inflater.inflate(R.layout.fragment_data, container, false);
        initView(view);
        return view;
    }

    private void initView(View view) {
        final EditText password = (EditText) view.findViewById(R.id.data);
        Button login = (Button) view.findViewById(R.id.login_data);

        login.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                if (getActivity() instanceof DataFragmentCallBack) {
                    ((DataFragmentCallBack) getActivity()).
                        onLoginClick(password.getText().toString());
```

```
                    }
                }
            });
        }

        public interface DataFragmentCallBack {
            void onLoginClick(String data);
        }
    }
```

让 Activity 实现 Fragment 中的接口，并实现该方法。这样每次在 Fragment 中处理的数据就都可以在 Activity 中获取并处理了。运行程序后效果如图 5-11 所示，接收数据后打印的 Log 如图 5-12 所示。



图 5-11　Fragment 向 Activity 传递数据

a.fragment I/DataActivity: lizishule

图 5-12　从 Activity 获取 Fragment 中数据

这种方法不但实现了 Fragment 向 Activity 传输数据，而且 Activity 与 Fragment 之间的耦合度非常低，任何一个实现了该接口的 Activity 都可以接收 Fragment 中的数据。

Activity 向 Fragment 传递数据更加简单，方法也很多。在 Activity 中使用 Fragment 时必须实例化 Fragment，在这个过程中可以传递数据。Google 官方不推荐使用构造函数传递参数，这里使用静态的 newInstance(Bundle bundle)方法。代码如下：

```
public class DataFragment extends Fragment {

    private static Bundle activityArgs;

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        View view = inflater.inflate(R.layout.fragment_data, container, false);
        initView(view);
        return view;
    }

    private void initView(View view) {
        final EditText password = (EditText) view.findViewById(R.id.data);
        Button login = (Button) view.findViewById(R.id.login_data);
        String name = activityArgs.getString("name");
        password.setText(name);
```

```
        }

        public static DataFragment newInstance(Bundle bundle) {
            activityArgs = bundle;
            DataFragment fragment = new DataFragment();
            return fragment;
        }
    }
```

在 Fragment 类中创建 newInstance(Bundle bundle)方法，Activity 在使用 Fragment 时调用此方法并传递 Bundle 参数，在初始化 Fragment 界面时将数据显示到界面上。其中，Activity 代码如下：

```
    public class DataActivity extends AppCompatActivity implements {

        @Override
        protected void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            setContentView(R.layout.activity_data);
            Bundle bundle = new Bundle();
            bundle.putString("name", "作者  李瑞奇");
            getSupportFragmentManager().beginTransaction().
                    replace(R.id.data, DataFragment.newInstance(bundle)).commit();
        }
    }
```

运行程序，显示效果如图 5-13 所示。

在 Fragment 中还有很多值得探究的问题，这里限于篇幅就不再继续讨论了。读者在使用过程中只要多加练习、仔细思考其中的原理，慢慢地就能够举一反三了。



图 5-13　从 Activity 向 Fragment 传递数据

5.6 小　结

本章系统地讲述了 Fragment 的使用场景、使用方法，讲解了 Fragment 的生命周期，并将其与 Activity 的生命周期做了比较，以便加深对 Fragment 的理解。同时对 ListFragment 与 DialogFragment 这两个特殊的 Fragment 进行了深入的讲解，对其用法和特性也都进行了分析。在本章最后还根据开发中的经验向读者阐释了一些常见的问题。



第 6 章 更多的控件与控件开发

本书在第 4 章介绍了 Android 的基本控件，这些控件已经能够帮助我们在开发过程中实现各种需要的 UI 设计。但是这些控件难免会不足。Google 公司出于优化的考虑，之后又陆续提供了一些新控件。本章将讲解这些新控件，以及如何去开发新控件。

6.1　ViewPager 的使用

ViewPager 是一个非常强大的 UI 组件，应用非常广泛。它提供了多界面切换的效果，具体来说就是：当前显示一组界面中的一个界面，当用户左右滑动界面时，当前的屏幕显示当前界面和下一个界面的一部分，滑动结束后，界面自动跳转到当前选择的界面中。

6.1.1　ViewPager 的使用

下面用实例来说明如何使用。

和之前的其他控件一样，使用 ViewPager 也需要把 ViewPager 对应的控件加入布局文件中。在布局文件中添加 ViewPager 的代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".activity.MainActivity">

    <android.support.v4.view.ViewPager
        android:id="@+id/viewPager"
        android:layout_width="match_parent"
        android:layout_height="match_parent">
    </android.support.v4.view.ViewPager>
</RelativeLayout>
```

当完成布局文件之后，在 Activity 文件中根据 id 获取 ViewPager。与使用 ListView 相似，还需要一个适配器类来处理 ViewPager 的页面。

Activity 代码如下：

```
package com.buaa.moreview.activity;

import android.support.v4.view.ViewPager;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;

import com.buaa.moreview.R;
import com.buaa.moreview.adapter.FirstViewPagerAdapter;

import java.util.ArrayList;
```

```
import java.util.List;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        initView();
    }

    private void initView() {
        ViewPager viewPager = (ViewPager) findViewById(R.id.viewPager);
        LayoutInflater layoutInflater = getLayoutInflater();

        View first = layoutInflater.inflate(R.layout.first_page, null);
        View second = layoutInflater.inflate(R.layout.second_page, null);
        View third = layoutInflater.inflate(R.layout.third_page, null);

        List<View> list = new ArrayList<>();
        list.add(first);
        list.add(second);
        list.add(third);

        FirstViewPagerAdapter firstViewPagerAdapter = new FirstViewPagerAdapter(list);
        viewPager.setAdapter(firstViewPagerAdapter);
    }
}
```

适配器代码如下：

```
package com.buaa.moreview.adapter;

import android.support.v4.view.PagerAdapter;
import android.view.View;
import android.view.ViewGroup;

import java.util.List;

public class FirstViewPagerAdapter extends PagerAdapter {
    private List<View> list;

    public FirstViewPagerAdapter(List<View> list) {
        this.list = list;
```

```
        }

        @Override
        public int getCount() {
            return list.size();
        }

        @Override
        public boolean isViewFromObject(View view, Object object) {
            return view == object;
        }

        @Override
        public Object instantiateItem(ViewGroup container, int position) {
            container.addView(list.get(position));
            return list.get(position);
        }

        @Override
        public void destroyItem(ViewGroup container, int position, Object object) {
            container.removeView(list.get(position));
        }
    }
```

在 Activity 中还涉及 3 个 View 页面。其布局如下（三者相同，只举其一）：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="first page by liruiqi"
        android:textSize="28sp" />
</LinearLayout>
```

运行程序，然后滑动页面，当页面滑动到一半时效果如图 6-1 所示，当滑动完成时效果如图 6-2 所示。

在上例中，主要部分有两处，一处是 initView()方法，一处是适配器类。

在 initView()方法中通过 findViewById()方法获取了 ViewPager，然后通过 LayoutInflater 类获取布局文件，使用 list 存储这 3 个布局文件对应的 View，再将 list 传入 FirstViewPagerAdapter 这个适配器中，再用 ViewPager 类的 setAdapter()方法让 FirstViewPagerAdapter 设置为它的适配

器。这个过程和 ListView 的过程非常相似，读者应该很容易理解。

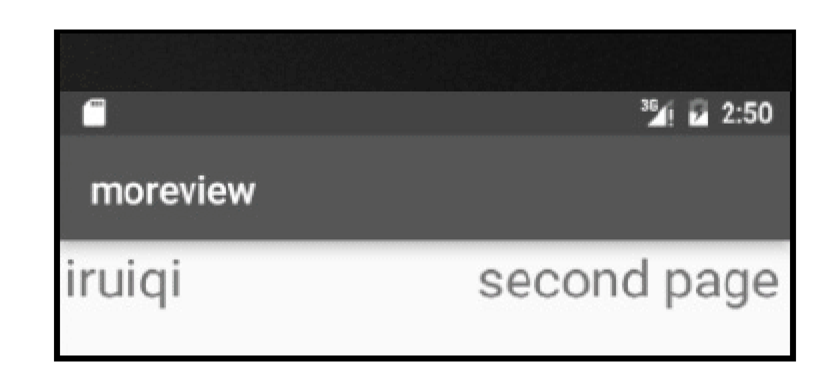


图 6-1　使用 ViewPager 时从第一页滑动到第二页的过程效果图

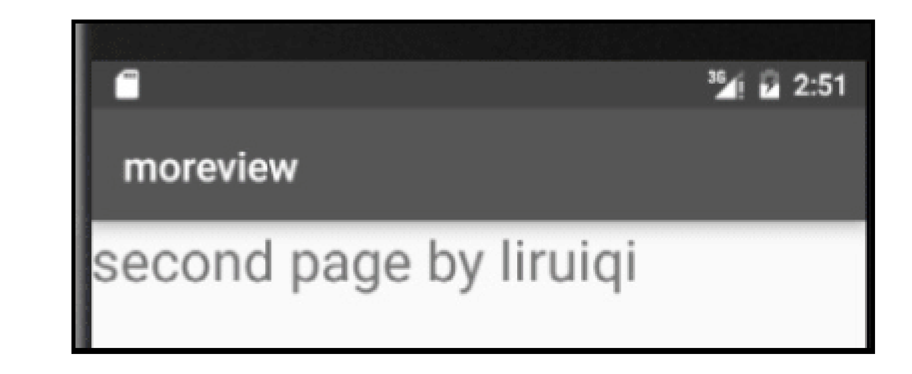


图 6-2　使用 ViewPager 时滑动到第二页时的效果

对于适配器，想必读者都不陌生，在 ListView 中也有适配器，listView 通过重写 getView()函数来获取当前要加载的 item，但是这里面的适配器 FirstViewPagerAdapter 类和 ListView 的适配器可能不太相同。

通过借助上面的实例，我们知道要实现一个 PagerAdapter 适配器至少需要覆盖表 6-1 中的 4 个方法。

表6-1　实现一个PagerAdapter适配器需要覆盖的方法

方法	作用
getCount()	返回要滑动的 View 个数
instantiateItem(ViewGroup container, int position)	创建指定位置的页面视图。适配器有责任增加即将创建的 View 视图到给定的 container 中，这是为了确保在 finishUpdate(viewGroup)返回时已经完成。此方法的返回值是一个代表新增视图页面的 Object（Key），上述实例中直接返回了视图本身。当然这里没必要非要返回视图本身，也可以是这个页面的其他容器，甚至是可以代表当前页面的任意值，只要可以与增加的 View 一一对应即可，比如 position 变量也可以作为 Key
destroyItem(ViewGroup container, int position, Object object)	移除一个给定位置的页面。适配器有责任从容器中删除这个视图。这是为了确保在 finishUpdate(viewGroup)返回时视图能够被移除
isViewFromObject(View, Object)	用来判断 instantiateItem(ViewGroup, int)方法所返回的 Key 与一个页面视图代表的是否为同一个视图（判断两者是否对应，对应就表示同一个 View）。本实例中 instantiateItem(ViewGroup, int)方法返回了视图本身，因为在此方法中使用视图与之比较，所以自然返回的是 true

另外，PagerAdapter 支持数据集合的改变，数据集合的改变必须要在主线程里面执行，然后还要调用 notifyDataSetChanged()方法。和 BaseAdapter 非常相似。数据集合的改变包括页面的添加删除和修改位置。

6.1.2　ViewPager 与 Fragment

在实际的开发过程中，ViewPager 与 Fragment 组合使用是比较常见的，而对于 fragment，它所使用的适配器是 FragmentPagerAdapter。FragmentPagerAdapter 继承自 PagerAdapter 类，用于呈现 Fragment 页面。这些 Fragment 页面会一直保存在 FragmentManager 中，以便用户随时取用。

这个适配器最好用于有限个静态 fragment 页面的管理。尽管不可见的视图有时会被销毁，但用户所有访问过的 fragment 都会被保存在内存中。因此 fragment 实例会保存大量的各种状态，这就造成了很大的内存开销。如果要处理大量的页面切换，就可以使用 FragmentStatePager-Adapter。

每一个使用 FragmentPagerAdapter 的 ViewPager 都要有一个有效的 ID 集合，有效 ID 的集合就是 Fragment 的集合。对于 FragmentPagerAdapter 的子类，只需要重写 getItem(int position)和 getCount()就可以了。

创建一个继承 FragmentPagerAdapter 类的类，代码如下：

```
package com.buaa.moreview.adapter;

import android.support.v4.app.Fragment;
import android.support.v4.app.FragmentManager;
import android.support.v4.app.FragmentPagerAdapter;

import java.util.List;

public class FragmentAdapter extends FragmentPagerAdapter {

    private List<Fragment> fragmentList;

    public FragmentAdapter(FragmentManager fm, List<Fragment> fragmentList) {
        super(fm);
        this.fragmentList = fragmentList;
    }

    @Override
    public Fragment getItem(int position) {
        return fragmentList.get(position);
    }

    @Override
    public int getCount() {
        return fragmentList.size();
    }
}
```

重写的两个方法的作用很明显：getItem(int position) 是根据位置获取 Fragment，getCount()是用来获取列表数量的。

下面创建 3 个 Fragment 类，代码如下：

```
public class FirstFragment extends Fragment {
```

```
        @Override
        public View onCreateView(LayoutInflater inflater, ViewGroup container,
                                 Bundle savedInstanceState) {
            return inflater.inflate(R.layout.fragment_first, container, false);
        }
    }
```

其他两个 Fragment 类与之没有区别，对应的布局文件只添加了一个文本框，展示一行文字。

在 Activity 类中调用适配器以及 Fragment 类，并将之与 ViewPager 进行整合。代码与展示 3 个普通页面的代码没有实质区别，用下面的方法替代 initView()方法即可。

```
private void initViewForFragment() {
    ViewPager viewPager = (ViewPager) findViewById(R.id.viewPager);

    FragmentManager fragmentManager = getSupportFragmentManager();

    List<Fragment> fragments = new ArrayList<Fragment>();
    fragments.add(new FirstFragment());
    fragments.add(new SecondFragment());
    fragments.add(new ThirdFragment());

    FragmentAdapter framentAdapter = new FragmentAdapter(fragmentManager, fragments);
    viewPager.setAdapter(framentAdapter);
}
```

运行应用的效果与之前的实例效果是一样的，只是此时使用的是 Fragment。在 Fragment 中可以做各种操作，如保存用户数据，这些都是单纯的页面做不到的。使用 Fragment 的优势还有很多，在第 5 章已经充分讲解，此处不再详述。

6.1.3 ViewPager 与 TabLayout

使用 TabLayout 很容易实现选项卡的功能。这里将结合 ViewPager、Fragment、TabLayout 实现一个具有选项卡功能的程序。

要实现这样一个程序，首先需要在布局文件中加入 TabLayout。直接加入到布局文件中会提示有错误，这是因为使用 TabLayout 需要在 build.gradle 文件中加入“compile 'com.android.support:design:23.4.0'”。这里的 23.4.0 代表使用的版本号，可以使用与支持包的版本号一致的版本，比如：

```
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    testCompile 'junit:junit:4.12'
    compile 'com.android.support:support-v4:23.4.0'
    compile 'com.android.support:design:23.4.0'
}
```

这时，再次在布局文件中加入 TabLayout 就可以使用了，具体如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:orientation="vertical"
    tools:context=".activity.MainActivity">

    <android.support.design.widget.TabLayout
        android:id="@+id/tab"
        app:tabMode="fixed"
        android:layout_width="match_parent"
        android:layout_height="wrap_content">
    </android.support.design.widget.TabLayout>
    <android.support.v4.view.ViewPager
        android:id="@+id/viewPager"
        android:layout_width="match_parent"
        android:layout_height="match_parent">
    </android.support.v4.view.ViewPager>
</LinearLayout>
```

完成这些后，3 个 Fragment 类文件不变，修改适配器类：

```
public class FragmentAdapter extends FragmentPagerAdapter {

    private List<Fragment> fragmentList;
    private List<String> tabList;

    public FragmentAdapter(FragmentManager fm, List<Fragment> fragmentList, List<String> tabList) {
        super(fm);
        this.fragmentList = fragmentList;
        this.tabList = tabList;
    }

    @Override
    public Fragment getItem(int position) {
        return fragmentList.get(position);
    }

    @Override
    public int getCount() {
        return fragmentList.size();
    }
```

```
        @Override
        public CharSequence getPageTitle(int position) {
            return tabList.get(position);
        }
    }
```

通过观察修改之后的适配器类可以发现其实就是加入了一个存储选项卡内容的列表以及一个获取选项卡内容的方法。在 Activity 类中进行调用，只需修改 initViewForFragment()方法即可：

```
    private void initViewForFragment() {
        FragmentManager fragmentManager = getSupportFragmentManager();
        List<Fragment> fragments = new ArrayList<Fragment>();
        fragments.add(new FirstFragment());
        fragments.add(new SecondFragment());
        fragments.add(new ThirdFragment());

        List<String> tabs = new ArrayList<>();
        tabs.add("首页");
        tabs.add("新闻");
        tabs.add("个人中心");
        FragmentAdapter framentAdapter = new FragmentAdapter(fragmentManager, fragments, tabs);

        ViewPager viewPager = (ViewPager) findViewById(R.id.viewPager);
        viewPager.setAdapter(framentAdapter);
        TabLayout tabLayout = (TabLayout) findViewById(R.id.tab);
        tabLayout.setupWithViewPager(viewPager);
    }
```

此时运行程序就出现了选项卡功能，不管是点击选项卡标签还是滑动界面都可以进行界面切换，效果如图 6-3 所以。



图 6-3 使用 ViewPager 与 TabLayout 一起进行布局时的效果

本实例与之前的实例区别在于程序的最后多了一个步骤。TabLayout 调用 setupWithViewPager(ViewPager viewPager)方法与 ViewPager 方法进行了绑定。具体来说就是在这个方法内，使用 viewPager 调用 getAdapter()方法获取了 vierPager 的适配器，然后调用 setPagerAdapter(adapter, true)方法设置了适配器。

此处需要提醒读者的是，在某些书籍中，可能 tabLayout.setupWithViewPager(viewPager)之后还有 tabLayout.setTabsFromPagerAdapter(framentAdapter)这样一行代码。这是因为在较早的版本中必须使用该方法才能设置适配器。但是该方法有较多缺陷，Google 公司将之废除了，并在 setupWithViewPager(ViewPager viewPager)方法中加入了执行设置适配器的功能。

这种方式是很多人喜欢使用的一种开发选项卡的方式，但是建议读者还是使用自定义的方式来进行开发选项卡：

（1）在布局文件中引入 TabLayout。

（2）在 Activity 中获取 TabLayout，并设置 tab 以及监听事件。在监听事件中设置点击某个 tab 时进入某个对应索引的 ViewPager 页面或者 Fragment。

（3）设置 ViewPager 的监听事件，当滑动切换到某个页面或者 Fragment 时选中对应索引的 tab。

在之前的实例上做上面的修改也是可以实现选项卡功能的，这里不再详细讲解，希望读者可以按照上述实现思路进行练习。

6.2 RecyclerView 的使用

RecyclerView 是一种比较新的控件，据官方的介绍，该控件用于在有限的窗口中展示大量数据集。其实具有这种功能的控件很多，读者并不陌生，比如 ListView、GridView。既然有了 ListView、GridView，为什么还需要 RecyclerView 这样的控件呢？从整体上看 RecyclerView 架构提供了一种插拔式的体验，高度解耦，异常灵活，通过设置它所提供的 LayoutManager、ItemDecoration、ItemAnimator，可以实现非常好的效果。我们可以通过导入 support-v7 使用 RecyclerView，并能通过布局管理器 LayoutManager 控制显示方式。如果想要控制 Item 间的间隔（可绘制），那么可以使用 ItemDecoration 类来控制。当然，还可以通过 ItemAnimator 类控制 Item 增删的动画（动画内容稍后几章会学习）。

6.2.1 RecyclerView 的实现

RecyclerView 与 ListView 的使用很像，也需要一个适配器以及一个 item 布局文件，还需要在 Activity 的布局文件中使用 RecyclerView。下面先用一个实例来说明如何使用 RecyclerView。

适配器类的代码如下：

```
package com.buaa.moreview.adapter;

import android.content.Context;
import android.support.v7.widget.RecyclerView;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.TextView;

import com.buaa.moreview.R;

import java.util.List;
```

```
import java.util.Map;

public class RecyclerAdapter extends RecyclerView.Adapter {
    private List<Map<String, String>> mapList;
    private Context context;

    public RecyclerAdapter(List<Map<String, String>> mapList, Context context) {
        this.mapList = mapList;
        this.context = context;
    }

    @Override
    public RecyclerView.ViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
        MyViewHolder myViewHolder = new MyViewHolder(LayoutInflater.from(
                context).inflate(R.layout.recycle_item, parent,
                false));
        return myViewHolder;
    }

    @Override
    public void onBindViewHolder(RecyclerView.ViewHolder holder, int position) {
        Map<String, String> map = mapList.get(position);
        MyViewHolder myViewHolder = (MyViewHolder) holder;
        myViewHolder.name.setText(map.get("name"));
        myViewHolder.age.setText(map.get("age"));
    }

    @Override
    public int getItemCount() {
        return mapList.size();
    }

    class MyViewHolder extends RecyclerView.ViewHolder {
        TextView name;
        TextView age;

        public MyViewHolder(View view) {
            super(view);
            name = (TextView) view.findViewById(R.id.name);
            age = (TextView) view.findViewById(R.id.age);
        }
    }
}
```

Item 的布局文件如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:orientation="horizontal">

    <TextView
        android:id="@+id/name"
        android:layout_height="wrap_content"
        android:layout_width="0dp"
        android:layout_weight="1"
        android:textSize="25sp" />

    <TextView
        android:id="@+id/age"
        android:layout_height="wrap_content"
        android:layout_width="0dp"
        android:layout_weight="1"
        android:textSize="25sp" />
</LinearLayout>
```

在 Activity 对应的布局文件中加入 RecyclerView：

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.buaa.moreview.activity.RecyclerViewActivity">

    <android.support.v7.widget.RecyclerView
        android:id="@+id/recycle"
        android:layout_width="match_parent"
        android:layout_height="match_parent"></android.support.v7.widget.RecyclerView>
</RelativeLayout>
```

上面的几个步骤看起来和 ListView 非常相似，理解了 ListView 也就很容易理解上述代码了。读者重点关注的应该是 Activity 部分的内容，代码如下：

```
package com.buaa.moreview.activity;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
```

```
import android.support.v7.widget.GridLayoutManager;
import android.support.v7.widget.LinearLayoutManager;
import android.support.v7.widget.RecyclerView;

import com.buaa.moreview.R;
import com.buaa.moreview.adapter.RecyclerAdapter;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

public class RecyclerViewActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_recycler_view);
        initView();
    }

    private void initView() {
        List<Map<String, String>> mapList = new ArrayList<>();
        Map<String, String> map;
        for (int i = 0; i < 10; i++) {
            map = new HashMap<>();
            map.put("name", "姓名为：" + i);
            map.put("age", "年龄为：" + i);
            mapList.add(map);
        }
        RecyclerAdapter recyclerAdapter = new RecyclerAdapter(mapList, RecyclerViewActivity.this);
        RecyclerView recyclerView = (RecyclerView) findViewById(R.id.recycle);
        recyclerView.setLayoutManager(new LinearLayoutManager(this));
        recyclerView.setAdapter(recyclerAdapter);
    }
}
```

细心的读者会发现这与 ListView 代码大同小异，只是多了 recyclerView.setLayoutManager (new LinearLayoutManager (this))代码。这行代码设置了数据该以什么样的布局文件展示。这是 RecyclerView 非常不一样的一个地方，通过设置可以在不改变其他内容的情况下改变 RecyclerView 为自己想要的样式。这种设计的耦合度是非常低的。

运行当前应用，效果如图 6-4 所示。



图 6-4　使用 RecyclerView 实现类似 ListView 的效果

下面我们把 recyclerView.setLayoutManager(new LinearLayoutManager(this))代码修改为 recyclerView.setLayout- Manager(new GridLayoutManager(this,3))，此处的参数 3 指的是 3 列。这里稍微修改一下布局，将 item 的布局文件修改如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:orientation="vertical">

    <TextView
        android:id="@+id/name"
        android:layout_width="wrap_content"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:textSize="25sp" />

    <TextView
        android:id="@+id/age"
        android:layout_width="wrap_content"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:textSize="25sp" />
</LinearLayout>
```

运行应用程序，就会变成图 6-5 所示的这种网格布局。

简单的代码变动就能很简单、很优雅地将布局从类似 ListView 的样式改变为类似于网格的布局，同时还不需要改变其他重要部分代码。与 GridLayoutManager（this，3）类似，如果想要实现瀑布流效果，直接将之更换为 StaggeredGridLayoutManager(3, StaggeredGridLayoutManager. VERTICAL)即可，其中的参数 StaggeredGridLayoutManager. VERTICAL 意味着竖向排列，此时的参数 3 代表 3 列，当然如果横向排列，那么参数 3 就代表 3 行了。

moreview
姓名为：0 姓名为：1 姓名为：2
年龄为：0 年龄为：1 年龄为：2
姓名为：3 姓名为：4 姓名为：5
年龄为：3 年龄为：4 年龄为：5
姓名为：6 姓名为：7 姓名为：8
年龄为：6 年龄为：7 年龄为：8
姓名为：9
年龄为：9

图 6-5　使用 RecyclerView 实现网格布局

6.2.2 item 分隔线及动画效果

上面的布局看起来有些别扭，为什么 item 间没有分隔线？其实 RecyclerView 并没有支持 divider 这样的属性。RecyclerView 允许我们自由地去定制它。分隔线是可以定制的，但是自定义的方式对开发者的水平要求较高，给 item 的布局去设置 margin 的方式更加常用，主要是因为更加简单。但是有时会需要更加复杂的分隔线，不得不使用自定义的方式，因此此处给出自定义实现的实例。

RecyclerView 用于加入分隔线的方法是 addItemDecoration()。该方法的参数为 RecyclerView.ItemDecoration——抽象类，官方目前并没有提供默认的实现类。下面以网格布局的分隔线为例提供一个实现类：

```
package com.buaa.moreview.decoration;

import android.content.Context;
import android.content.res.TypedArray;
import android.graphics.Canvas;
import android.graphics.Rect;
import android.graphics.drawable.Drawable;
import android.support.v7.widget.GridLayoutManager;
import android.support.v7.widget.RecyclerView;
import android.support.v7.widget.RecyclerView.LayoutManager;
import android.support.v7.widget.RecyclerView.State;
import android.support.v7.widget.StaggeredGridLayoutManager;
import android.view.View;

public class GridItemDecoration extends RecyclerView.ItemDecoration {

    private static final int[] ATTRS = new int[]{android.R.attr.listDivider};
    private Drawable mDivider;

    public GridItemDecoration(Context context) {
        final TypedArray a = context.obtainStyledAttributes(ATTRS);
        mDivider = a.getDrawable(0);
        a.recycle();
    }

    @Override
    public void onDraw(Canvas c, RecyclerView parent, State state) {
        drawHorizontal(c, parent);
        drawVertical(c, parent);
    }
```

```
private int getSpanCount(RecyclerView parent) {
    int spanCount = -1;
    LayoutManager layoutManager = parent.getLayoutManager();
    if (layoutManager instanceof GridLayoutManager) {

        spanCount = ((GridLayoutManager) layoutManager).getSpanCount();
    } else if (layoutManager instanceof StaggeredGridLayoutManager) {
        spanCount = ((StaggeredGridLayoutManager) layoutManager)
                .getSpanCount();
    }
    return spanCount;
}

public void drawHorizontal(Canvas c, RecyclerView parent) {
    int childCount = parent.getChildCount();
    for (int i = 0; i < childCount; i++) {
        final View child = parent.getChildAt(i);
        final RecyclerView.LayoutParams params = (RecyclerView.LayoutParams) child
                .getLayoutParams();
        final int left = child.getLeft() - params.leftMargin;
        final int right = child.getRight() + params.rightMargin
                + mDivider.getIntrinsicWidth();
        final int top = child.getBottom() + params.bottomMargin;
        final int bottom = top + mDivider.getIntrinsicHeight();
        mDivider.setBounds(left, top, right, bottom);
        mDivider.draw(c);
    }
}

public void drawVertical(Canvas c, RecyclerView parent) {
    final int childCount = parent.getChildCount();
    for (int i = 0; i < childCount; i++) {
        final View child = parent.getChildAt(i);

        final RecyclerView.LayoutParams params = (RecyclerView.LayoutParams) child
                .getLayoutParams();
        final int top = child.getTop() - params.topMargin;
        final int bottom = child.getBottom() + params.bottomMargin;
        final int left = child.getRight() + params.rightMargin;
        final int right = left + mDivider.getIntrinsicWidth();

        mDivider.setBounds(left, top, right, bottom);
        mDivider.draw(c);
```

```
        }
    }

    private boolean isLastColum(RecyclerView parent, int pos, int spanCount,
                                int childCount) {
        LayoutManager layoutManager = parent.getLayoutManager();
        if (layoutManager instanceof GridLayoutManager) {
            if ((pos + 1) % spanCount == 0) {
                return true;
            }
        } else if (layoutManager instanceof StaggeredGridLayoutManager) {
            int orientation = ((StaggeredGridLayoutManager) layoutManager)
                    .getOrientation();
            if (orientation == StaggeredGridLayoutManager.VERTICAL) {
                if ((pos + 1) % spanCount == 0) {
                    return true;
                }
            } else {
                childCount = childCount - childCount % spanCount;
                if (pos >= childCount)
                   return true;
            }
        }
        return false;
    }

    private boolean isLastRaw(RecyclerView parent, int pos, int spanCount,
                              int childCount) {
        LayoutManager layoutManager = parent.getLayoutManager();
        if (layoutManager instanceof GridLayoutManager) {
            childCount = childCount - childCount % spanCount;
            if (pos >= childCount)
                return true;
        } else if (layoutManager instanceof StaggeredGridLayoutManager) {
            int orientation = ((StaggeredGridLayoutManager) layoutManager)
                    .getOrientation();
            if (orientation == StaggeredGridLayoutManager.VERTICAL) {
                childCount = childCount - childCount % spanCount;
                if (pos >= childCount)
                    return true;
            } else {
                if ((pos + 1) % spanCount == 0) {
                    return true;
```

```
                }
            }
        }
        return false;
    }

    @Override
    public void getItemOffsets(Rect outRect, int itemPosition,
                               RecyclerView parent) {
        int spanCount = getSpanCount(parent);
        int childCount = parent.getAdapter().getItemCount();
        if (isLastRaw(parent, itemPosition, spanCount, childCount)) {
            outRect.set(0, 0, mDivider.getIntrinsicWidth(), 0);
        } else if (isLastColum(parent, itemPosition, spanCount, childCount)) {
            outRect.set(0, 0, 0, mDivider.getIntrinsicHeight());
        } else {
            outRect.set(0, 0, mDivider.getIntrinsicWidth(),
                    mDivider.getIntrinsicHeight());
        }
    }
}
```

当我们在 Activity 类的 initView 方法中加入 recyclerView.addItemDecoration(new GridItemDecoration(this))之后，运行程序，效果如图 6-6 所示，实现了分隔线效果。

此处实现的分隔线是针对 GridLayoutManager 与 StaggeredGridLayoutManager 的。如果要使用 LinearLayoutManager，读者可以参照上例自行开发。

图 6-6　RecyclerView 实现分隔线以及动画效果

关于动画的使用更加简单，只需在 initView 方法中加入 recyclerView.setItemAnimator(new DefaultItemAnimator())即可。此处的 DefaultItemAnimator 类是官方提供的默认动画效果。此时，在界面中加入一个删除按钮，每点击一次就删除一个 item。在 RecyclerView 中删除 item 需要在适配器类中增加一个 removeData()方法，并在按钮的点击事件中调用这个方法，而不是像 ListView 那样使用 adapter.notifyDataSetChanged()方法。removeData 方法的代码如下：

```
public void removeData(int position) {
    mapList.remove(position);
    notifyItemRemoved(position);
}
```

运行程序，可以发现动画效果是很明显的。除了默认的动画效果类之外，官方还提供了 SlideInOutLeftItemAnimator、SlideInOutRightItemAnimator、SlideInOutTopItemAnimator、

SlideInOutBottomItemAnimator 等动画效果，读者可以多做实验，观察其中的效果。

通过本实例可以看出使用 RecyclerView 的优点确实很多，开发者可以通过调用一个方法简单地实现布局格式、动画效果、分隔线等的切换，做到了解耦合，实现了热插拔的效果。

6.2.3 点击事件的实现

官方并没有给 RecyclerView 提供 ClickListener 和 LongClickListener，需要开发者自己去添加。添加点击事件的代码并不复杂，在 RecyclerAdapter 类中加入并修改 onBindViewHolder() 方法即可：

```
private OnItemClickLitener mOnItemClickLitener;

public void setOnItemClickLitener(OnItemClickLitener mOnItemClickLitener)
{
      this.mOnItemClickLitener = mOnItemClickLitener;
}
public interface OnItemClickLitener
{
      void onItemClick(View view, int position);
      void onItemLongClick(View view , int position);
}
@Override
public void onBindViewHolder(RecyclerView.ViewHolder holder, int position) {
      Map<String, String> map = mapList.get(position);
      final MyViewHolder myViewHolder = (MyViewHolder) holder;
      myViewHolder.name.setText(map.get("name"));
      myViewHolder.age.setText(map.get("age"));

      if (mOnItemClickLitener != null)
      {
            myViewHolder.itemView.setOnClickListener(new View.OnClickListener()
            {
                  @Override
                  public void onClick(View v)
                  {
                        int pos = myViewHolder.getLayoutPosition();
                        mOnItemClickLitener.onItemClick(myViewHolder.itemView, pos);
                  }
            });

            myViewHolder.itemView.setOnLongClickListener(new View.OnLongClickListener()
            {
                  @Override
```

```
                    public boolean onLongClick(View v)
                    {
                        int pos = myViewHolder.getLayoutPosition();
                        mOnItemClickLitener.onItemLongClick(myViewHolder.itemView, pos);
                        return false;
                    }
                });
            }
        }
```

简单地说就是在 RecyclerAdapter 类中加入了一个监听事件的接口，并在 onBindViewHolder()方法中调用接口；增加了 setOnItemClickLitener()方法，它的参数就是监听事件的接口，当 RecyclerAdapter 适配器类调用 setOnItemClickLitener()方法时必须实现这个接口。在 Activity 类中使用 RecyclerAdapter 类调用接口 setOnItemClickLitener()方法的代码如下：

```
recyclerAdapter.setOnItemClickLitener(new RecyclerAdapter.OnItemClickLitener() {
    @Override
    public void onItemClick(View view, int position) {
        Toast.makeText(RecyclerViewActivity.this,"点击事件",Toast.LENGTH_LONG).show();
    }

    @Override
    public void onItemLongClick(View view, int position) {
        Toast.makeText(RecyclerViewActivity.this,"长按事件",Toast.LENGTH_LONG).show();
    }
});
```

整个过程很容易理解，不再详细阐述过程。运行程序，点击 item 后，效果如图 6-7 所示，长按 item 的效果如图 6-8 所示。



图 6-7　RecyclerView 的 item 点击事件



图 6-8　RecyclerView 的 item 长按事件

6.3 自定义 View 控件

虽然官方在推出新版本时一般都会提供一些新的控件，但是在开发过程中有时会需要一些特别的控件，这时就需要自定义 View。自定义 View 的实现方式大概可以分为 3 种，即自绘控件、组合控件以及继承控件。下面就依次来讲解每种方式是如何自定义 View 的。

6.3.1 自绘控件

自绘控件的意思就是这个 View 上所展现的内容全部都是我们自己绘制出来的，但是想要完全自行绘制控件，大家可能还是一头雾水。在 Android 中，任何一个布局、任何一个控件其实都是直接或间接继承自 View 的，如 TextView、Button、ImageView、ListView 等。这些控件虽然是 Android 系统本身就提供好的，任何一个视图都不可能凭空突然出现在屏幕上，它们都是要经过非常科学的绘制流程后才能显示出来的。每一个视图的绘制过程都必须经历 3 个最主要的阶段，即 onMeasure()、onLayout()和 onDraw()。所以在自绘控件执行时，我们要先了解绘制控件的 3 个主要方法。

（1）onMeasure()。measure 是测量的意思，顾名思义 onMeasure()方法就是用于测量视图大小的。View 系统的绘制流程会从 ViewRoot 的 performTraversals()方法中开始，在其内部调用 View 的 measure()方法。measure()方法接收两个参数，即 widthMeasureSpec 和 heightMeasureSpec，分别用于确定视图的宽度和高度的规格和大小。

（2）onLayout()。measure 过程结束后，视图的大小就测量好了，接下来就是 layout 的过程了。正如其名字所描述的那样，这个方法是用于给视图进行布局的，也就是确定视图的位置。ViewRoot 的 performTraversals()方法会在 measure 结束后继续执行，并调用 View 的 layout()方法来执行此过程。layout()方法接收 4 个参数，分别代表左、上、右、下坐标，当然这个坐标是相对于当前视图的父视图而言的。可以看到，这里还把刚才测量出的宽度和高度传到了 layout()方法中。

（3）onDraw()。measure 和 layout 的过程都结束后，接下来进入到 draw 的过程。同样，根据名字可以判断出在这里才真正开始对视图进行绘制。ViewRoot 中的代码会继续执行并创建出一个 Canvas 对象，然后调用 View 的 draw()方法来执行具体的绘制工作。View 是不会帮我们绘制内容部分的，因此需要每个视图根据想要展示的内容来自行绘制。观察 TextView、ImageView 等类的源码，就会发现它们都重写了 onDraw()这个方法，并且在里面执行了相当多的绘制逻辑。

从上面对 3 个方法的分析来看，想要自绘控件，必须要重写的方法是 onDraw()方法，甚至说只要重写了 onDraw()方法就可以自绘控件了。下面通过一个实例来演示如何重写 onDraw()方法进行自绘控件，具体分三步来进行。

（1）自定义 View 的属性，首先在 res/values/下建立一个 attrs.xml，在里面定义 View 的属性。

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <attr name="ViewText" format="string" />
    <attr name="ViewTextSize" format="dimension" />
    <declare-styleable name="MyView">
        <attr name="ViewText" />
        <attr name="ViewTextSize" />
    </declare-styleable>
</resources>
```

在文件中我们定义了字体、字体大小两个属性，其中 fromat 指定的是属性的类型。

（2）继承 View 类，实现自定义的控件类，代码如下：

```
package com.buaa.moreview.view;

import android.content.Context;
import android.content.res.TypedArray;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.Paint;
import android.graphics.Rect;
import android.util.AttributeSet;
import android.util.TypedValue;
import android.view.View;

import com.buaa.moreview.R;

public class MyView extends View implements View.OnClickListener {
    private Paint mPaint;
    private Rect mBounds;
    private String viewText;
    private int textSize;

    public MyView(Context context, AttributeSet attrs) {
        super(context, attrs);
        mPaint = new Paint(Paint.ANTI_ALIAS_FLAG);
        mBounds = new Rect();
        TypedArray typedArray = context.getTheme().
                obtainStyledAttributes(attrs, R.styleable.MyView, 0, 0);
        setAttrs(typedArray);
    }

    private void setAttrs(TypedArray typedArray) {
        int n = typedArray.getIndexCount();
```

```
                for (int i = 0; i < n; i++) {
                    int attr = typedArray.getIndex(i);
                    switch (attr) {
                        case R.styleable.MyView_ViewText:
                            viewText = typedArray.getString(attr);
                            break;
                        case R.styleable.MyView_ViewTextSize:
                            textSize = typedArray.getDimensionPixelSize(attr,
                                    (int) TypedValue.applyDimension(
                                    TypedValue.COMPLEX_UNIT_SP,
                                            24,
                                            getResources().getDisplayMetrics()));
                            break;
                    }
                }
                typedArray.recycle();
        }

        @Override
        protected void onDraw(Canvas canvas) {
                super.onDraw(canvas);
                mPaint.setColor(Color.BLUE);
                canvas.drawRect(0, 0, getWidth(), getHeight(), mPaint);
                mPaint.setColor(Color.YELLOW);
                mPaint.setTextSize(textSize);
                String text = String.valueOf(viewText);
                mPaint.getTextBounds(text, 0, text.length(), mBounds);
                float textWidth = mBounds.width();
                float textHeight = mBounds.height();
                canvas.drawText(text, getWidth() / 2 - textWidth / 2, getHeight() / 2
                        + textHeight / 2, mPaint);
        }

        @Override
        public void onClick(View v) {
                v.setOnClickListener(this);
        }

}
```

在构造函数中用 context.getTheme().obtainStyledAttributes(attrs, R.styleable.MyView, 0, 0)来获取在 attrs 文件中 MyView 的属性列表，并在 setAttrs()方法中通过 TypedArray 对象获取属性的参数信息，并重写 onDraw(Canvas canvas)方法，根据这些参数、属性信息绘制出控件。同

时，让 MyView 类实现点击事件监听接口，使控件能够触发点击事件。

（3）在布局文件中声明 MyView：

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.buaa.moreview.activity.MyViewActivity">

    <com.buaa.moreview.view.MyView
        android:id="@+id/my_view"
        android:layout_width="200dp"
        android:layout_height="200dp"
        app:ViewText="自定义控件"
        app:ViewTextSize="30sp" />
</RelativeLayout>
```

向使用普通控件一样，将自定义的 MyView 控件加入布局文件中，按照在 attrs.xml 文件中定义的属性向布局文件中添加属性。

经过上面 3 个步骤，此时已经绘制出了一个全新的控件。为了验证自定义的控件的可用性，在 Activity 中加入一个点击事件：

```
public class MyViewActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_my_view);

        MyView myView = (MyView) findViewById(R.id.my_view);
        myView.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Toast.makeText(MyViewActivity.this,
        "第一个自定义控件", Toast.LENGTH_LONG).show();
            }
        });
    }
}
```

运行应用，效果如图 6-9 所示。

在实例中，自定义的控件在添加进布局文件时宽与高是指定的，显示正常。如果指定为

WRAP_CONTENT 将不能实现预期的效果。这是因为如果我们设置了明确的宽度和高度，系统测量的结果就是我们设置的结果；如果我们设置为 WRAP_CONTENT 或者 MATCH_PARENT，系统测量的结果就是 MATCH_PARENT 的长度。当设置了 WRAP_CONTENT 时，需要自己进行测量，即重写 onMesure()方法。



图 6-9 通过自绘控件来实现自定义控件的效果

重写之前先了解 MeasureSpec 的 specMode，一共包括 3 种类型：

（1）EXACTLY：表示父视图希望子视图的大小应该由 specSize 的值来决定，系统默认会按照这个规则设置子视图的大小，开发人员当然也可以按照自己的意愿设置成任意大小。

（2）AT_MOST：表示子视图最多只能是 specSize 中指定的大小，开发人员应该尽可能小地去设置这个视图，并且保证不会超过 specSize。系统默认会按照这个规则来设置子视图的大小，开发人员当然也可以按照自己的意愿设置成任意大小。

（3）UNSPECIFIED：表示开发人员可以按照自己的意愿将视图设置成任意大小，没有任何限制。这种情况比较少见，很少用到。

重写 onMeasure()方法的代码如下：

```
@Override
protected void onMeasure(int widthMeasureSpec, int heightMeasureSpec) {
    int widthMode = MeasureSpec.getMode(widthMeasureSpec);
    int widthSize = MeasureSpec.getSize(widthMeasureSpec);
    int heightMode = MeasureSpec.getMode(heightMeasureSpec);
    int heightSize = MeasureSpec.getSize(heightMeasureSpec);
    int width;
    int height;
    if (widthMode == MeasureSpec.EXACTLY) {
        width = widthSize;
    } else {
        mPaint.setTextSize(textSize);
        mPaint.getTextBounds(viewText, 0, viewText.length(), mBounds);
        float textWidth = mBounds.width();
        int desired = (int) (getPaddingLeft() + textWidth + getPaddingRight());
        width = desired;
    }

    if (heightMode == MeasureSpec.EXACTLY) {
        height = heightSize;
    } else {
        mPaint.setTextSize(textSize);
        mPaint.getTextBounds(viewText, 0, viewText.length(), mBounds);
```

```
            float textHeight = mBounds.height();
            int desired = (int) (getPaddingTop() + textHeight + getPaddingBottom());
            height = desired;
        }
        setMeasuredDimension(width, height);
    }
```

重写的 onMeasure()方法是标准的重写方式，读者以后要重写的话可以替换一部分参数，如 viewText 和 textSize，然后直接使用。在重写的方法中主要进行了一次判断，specMode 不是 EXACTLY 类型的都按照控件内容来计算控件需要的宽和高。

此时将控件的宽与高改为 WRAP_CONTENT，运行程序，效果就和我们预期的一致了，如图 6-10 所示。



图 6-10 修改宽高后的自绘控件

6.3.2 继承控件

继承控件的意思就是并不需要重头去实现一个控件，只需要继承一个现有的控件，然后在这个控件上增加一些新功能就可以形成一个自定义的控件。这种继承控件的特点就是不仅能够按照我们的需求加入相应的功能，还可以保留原生控件的所有功能。

为了能够加深读者对这种自定义 View 方式的理解，下面再来编写一个新的继承控件。相信每一个 Android 程序员都使用过 ListView，这次我们准备对 ListView 进行扩展，加入在 ListView 上滑动就可以显示一个删除按钮、点击按钮就会删除相应数据的功能。

首先准备一个删除按钮的布局，新建 delete_button.xml 文件，代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<Button xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:text="删除">
</Button>
```

这个布局文件很简单，只有一个按钮而已。接着创建 MyListView（自定义的 View）继承自 ListView，代码如下：

```
package com.buaa.moreview.view;

import android.content.Context;
import android.util.AttributeSet;
import android.view.GestureDetector;
import android.view.LayoutInflater;
import android.view.MotionEvent;
import android.view.View;
import android.view.ViewGroup;
```

```
import android.widget.ListView;
import android.widget.RelativeLayout;

import com.buaa.moreview.R;

public class MyListView extends ListView implements View.OnTouchListener,
        GestureDetector.OnGestureListener {

    private GestureDetector gestureDetector;
    private OnDeleteListener listener;
    private View deleteButton;
    private ViewGroup itemLayout;
    private int selectedItem;
    private boolean isDeleteShown;

    public MyListView(Context context, AttributeSet attrs) {
        super(context, attrs);
        gestureDetector = new GestureDetector(getContext(), this);
        setOnTouchListener(this);
    }

    public void setOnDeleteListener(OnDeleteListener l) {
        listener = l;
    }

    @Override
    public boolean onTouch(View v, MotionEvent event) {
        if (isDeleteShown) {
            itemLayout.removeView(deleteButton);
            deleteButton = null;
            isDeleteShown = false;
            return false;
        } else {
            return gestureDetector.onTouchEvent(event);
        }
    }

    @Override
    public boolean onDown(MotionEvent e) {
        if (!isDeleteShown) {
            selectedItem = pointToPosition((int) e.getX(), (int) e.getY());
        }
        return false;
```

```
    }

    @Override
    public boolean onFling(MotionEvent e1, MotionEvent e2, float velocityX,
                            float velocityY) {
        if (!isDeleteShown && Math.abs(velocityX) > Math.abs(velocityY)) {
            deleteButton = LayoutInflater.from(getContext()).inflate(
                    R.layout.detele_lay, null);
            deleteButton.setOnClickListener(new OnClickListener() {
                @Override
                public void onClick(View v) {
                    itemLayout.removeView(deleteButton);
                    deleteButton = null;
                    isDeleteShown = false;
                    listener.onDelete(selectedItem);
                }
            });
            itemLayout = (ViewGroup) getChildAt(selectedItem
                    - getFirstVisiblePosition());
            RelativeLayout.LayoutParams params = new RelativeLayout.LayoutParams(
                    LayoutParams.WRAP_CONTENT, LayoutParams.WRAP_CONTENT);
            params.addRule(RelativeLayout.ALIGN_PARENT_RIGHT);
            params.addRule(RelativeLayout.CENTER_VERTICAL);
            itemLayout.addView(deleteButton, params);
            isDeleteShown = true;
        }
        return false;
    }

    @Override
    public boolean onSingleTapUp(MotionEvent e) {
        return false;
    }

    @Override
    public void onShowPress(MotionEvent e) {
    }

    @Override
    public boolean onScroll(MotionEvent e1, MotionEvent e2, float distanceX,
                             float distanceY) {
        return false;
    }
```

```
        @Override
        public void onLongPress(MotionEvent e) {
        }

        public interface OnDeleteListener {
            void onDelete(int index);
        }
    }
```

上述代码在 MyListView 的构造方法中创建了一个 GestureDetector 的实例，用于监听手势，然后给 MyListView 注册了 touch 监听事件。然后在 onTouch()方法中进行判断，如果删除按钮已经显示就将其移除，如果删除按钮没有显示就使用 GestureDetector 来处理当前手势。

当手指按下时会调用 OnGestureListener 的 onDown()方法，在这里通过 pointToPosition()方法来判断当前选中的是 ListView 的哪一行。当手指快速滑动时会调用 onFling()方法，在这里会去加载 delete_button.xml 布局，然后将删除按钮添加到当前选中的那一行 item 上。注意，我们还给删除按钮添加了一个点击事件，当点击了删除按钮时就会回调 onDeleteListener 的 onDelete()方法，在回调方法中去处理具体的删除操作。

MyListView 类是本实例的核心部分，完成它之后就可以在 Activity 类中使用了，和普通的 ListView 没有什么区别。创建一个 item 文件，代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        android:id="@+id/my_list_text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
</LinearLayout>
```

然后创建一个适配器 MyAdapter，在这个适配器中加载 my_list_view_item 布局，代码如下：

```
package com.buaa.moreview.adapter;

import android.content.Context;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.BaseAdapter;
import android.widget.TextView;

import com.buaa.moreview.R;
```

```
import java.util.List;

public class MyListAdapter extends BaseAdapter {

    private List<String> list;
    private Context context;
    private LayoutInflater inflater;

    public MyListAdapter(Context context, List<String> list) {
        this.context = context;
        this.list = list;
        this.inflater = LayoutInflater.from(context);
    }


    @Override
    public int getCount() {
        return list.size();
    }

    @Override
    public Object getItem(int position) {
        return list.get(position);
    }

    @Override
    public long getItemId(int position) {
        return position;
    }

    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
        ViewHolder holder;
        if (convertView == null) {
            convertView = inflater.inflate(R.layout.list_item, null);
            holder = new ViewHolder();
            holder.text = (TextView) convertView.findViewById(R.id.my_list_text);
            convertView.setTag(holder);
        } else {
            holder = (ViewHolder) convertView.getTag();
        }
        holder.text.setText(list.get(position));
        return convertView;
```

```
        }

        static class ViewHolder {
            TextView text;
        }
}
```

最后在布局文件中加入自定义控件：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context="com.buaa.moreview.activity.MyListActivity">

    <com.buaa.moreview.view.MyListView
        android:id="@+id/my_list_view"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"></com.buaa.moreview.view.MyListView>
</LinearLayout>
```

并在 Activity 中使用自定义的 ListView，代码如下：

```
package com.buaa.moreview.activity;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;

import com.buaa.moreview.R;
import com.buaa.moreview.adapter.MyListAdapter;
import com.buaa.moreview.view.MyListView;

import java.util.ArrayList;
import java.util.List;

public class MyListActivity extends AppCompatActivity {

    private MyListView myListView;
    private List<String> list;
    private MyListAdapter myListAdapter;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
```

```
            super.onCreate(savedInstanceState);
            setContentView(R.layout.activity_my_list);
            initList();
            myListView = (MyListView) findViewById(R.id.my_list_view);
            myListAdapter = new MyListAdapter(this, list);
            myListView.setAdapter(myListAdapter);

            myListView.setOnDeleteListener(new MyListView.OnDeleteListener() {
                @Override
                public void onDelete(int index) {
                    list.remove(index);
                    myListAdapter.notifyDataSetChanged();
                }
            });
        }

        private void initList() {
            list = new ArrayList<>();
            for (int i = 0; i < 20; i++) {
                list.add("第" + (i + 1) + "行");
            }
        }
    }
```

完成实例之后，运行程序，会看到 MyListView 可以像 ListView 一样正常显示所有的数据，但是当你用手指在 MyListView 的某一行上快速滑动时会有一个删除按钮显示出来，如图 6-11 所示。

图 6-11　通过继承控件来实现自定义控件的效果

通过这个实例，读者应该已经明白通过继承控件来开发新控件的流程了。通过继承控件来增加新功能既可以使用原有功能，也可以使用新功能，是在开发实践中常用的方式。

6.3.3　组合控件

不需要自己绘制视图上显示的内容，只是将几个系统原生的控件组合到一起而创建出来的控件就是组合控件。

这种定义控件的方式很简单，下面就用一个 TextView 控件、EditText 控件以及一个 Button 控件组合成一个新的用来登录的控件实例进行讲解。

创建一个 login.xml 的布局文件，然后在布局文件中将上述 3 种控件加入布局文件中，代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
```

```
        android:layout_width="match_parent"
        android:layout_height="match_parent">

        <TextView
            android:id="@+id/login_title"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:gravity="center"
            android:textSize="26sp"/>

        <EditText
            android:id="@+id/login_password"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:password="true"
            android:textSize="26sp"
            android:hint="请输入密码"
            android:layout_below="@id/login_title"/>

        <Button
            android:id="@+id/login_button"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_below="@id/login_password"/>
    </RelativeLayout>
```

在这个布局文件中定义了一个 TextView 作为标题，加入了一个 EditText 用于输入密码，还有一个 Button 用于点击登录，代码很简单。下面再创建一个继承自 FrameLayout 类的 LoginView 类：

```
package com.buaa.moreview.view;

import android.content.Context;
import android.util.AttributeSet;
import android.view.LayoutInflater;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.FrameLayout;
import android.widget.TextView;
import android.widget.Toast;

import com.buaa.moreview.R;
```

```
public class LoginView extends FrameLayout {
    private Button loginButton;
    private TextView titleText;
    private EditText editText;

    public LoginView(final Context context, AttributeSet attrs) {
        super(context, attrs);
        LayoutInflater.from(context).inflate(R.layout.login, this);
        titleText = (TextView) findViewById(R.id.login_title);
        editText = (EditText) findViewById(R.id.login_password);
        loginButton = (Button) findViewById(R.id.login_button);
        loginButton.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                Toast.makeText(context,
                        "欢迎登录" + titleText.getText().toString() +
                                ",您的密码是：" + editText.getText().toString(),
                        Toast.LENGTH_LONG).show();
            }
        });
    }

    public void setTitleText(String text) {
        titleText.setText(text);
    }

    public void setLoginButtonText(String text) {
        loginButton.setText(text);
    }

}
```

在这个类中调用了 LayoutInflater 的 inflate()方法来加载刚刚定义的 login.xml 布局，并使用 findViewById()方法获取了 login.xml 文件中的 3 个控件，并给 Button 按钮设置了点击事件。同时，还建立了几个 set 方法，便于在 Activity 中使用此 View。

到此，组合控件 LoginView 就创建完成了。我们可以在布局文件中使用它，就像使用 TextView 等普通控件一样。在 activity_login.xml 文件中使用 LoginView，代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:gravity="center"
    tools:context="com.buaa.moreview.activity.LoginActivity">
```

```
    <com.buaa.moreview.view.LoginView
        android:id="@+id/login_page"
        android:layout_width="400dp"
        android:layout_height="300dp"></com.buaa.moreview.view.LoginView>
</LinearLayout>
```

此时只需要在 Activity 中初始化一些参数就可以使用了，Activity 中的代码如下：

```
public class LoginActivity extends AppCompatActivity {

    private LoginView loginView;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_login);

        loginView = (LoginView) findViewById(R.id.login_page);
        loginView.setTitleText("微博");
        loginView.setLoginButtonText("登录");
    }
}
```

运行程序，效果如图 6-12 所示。

组合控件的开发很简单，因为它是直接使用现有控件，在现有控件基础上进行的组合。但是组合控件也很有用，比如实例中的这个组合控件，将之优化之后就可以不停地复用，而不用每次遇到登录界面都要开发了。组合控件可以针对部分问题做到一次开发多次使用，所以希望读者多加重视。



图 6-12　通过组合控件来实现自定义控件的效果

6.4 小　　结

本章主要非常详细地讲述了 ViewPager、RecyclerView 这两个 View 控件的使用。这两个控件都是比较新的控件，在以往的 Android 开发书籍中没有讲解过，而在实际的开发过程中又经常使用，所以不得不使用大量篇幅去讲解。

同时，还对如何针对一些特殊情况（官方提供的控件不够或者不足以解决问题）自定义控件进行了讲解。自定义控件是一个 Android 工程师从初级向高级进发的必经之路，希望读者可以熟练地掌握不同自定义控件方式的技巧以及使用场景。



第 7 章 数据存储

任何一种开发都会涉及数据存储，在 Android 开发中数据存储更是操作频繁。在 Android 平台中实现数据存储有以下 5 种方式：

- 使用 SharedPreferences 存储数据。
- 使用文件存储数据。
- 使用 SQLite 数据库存储数据。
- 使用 ContentProvider 存储数据。
- 使用网络在云端存储数据。

网络存储方式将在之后的章节中讲解，本章重点讲述前 4 种数据存储方式。

7.1 SharedPreferences

SharedPreferences 经常被用于保存少量的数据，且这些数据的格式非常简单：字符串型、基本类型的值，比如应用程序的各种配置信息（是否打开音效、是否使用震动效果、小游戏的玩家积分等）、解锁口令密码等。

SharedPreferences 的原理是保存基于 XML 文件存储的 key-value 键值对数据，通常用来存储一些简单的配置信息。通过 Android Studio 中 DDMS 的 File Explorer 面板展开文件浏览树，很明显 SharedPreferences 数据总是存储在/data/data/<package name>/shared_prefs 目录下。

SharedPreferences 对象本身只能获取数据而不支持存储和修改，存储、修改是通过 SharedPreferences.edit()获取的内部接口 Editor 对象来实现的。SharedPreferences 本身是一个接口，程序无法直接创建 SharedPreferences 实例，只能通过 Context 提供的 getSharedPreferences(String name, int mode)方法来获取 SharedPreferences 实例，该方法中 name 表示要操作的 xml 文件名，第二个参数 mode 具体有下面 4 种形式：

- Context.MODE_PRIVATE：指定 SharedPreferences 数据只能被本应用程序读、写。
- Context.MODE_APPEND：检查文件是否存在，存在就向文件中追加内容，否则创建新文件。
- Context.MODE_WORLD_READABLE：指定 SharedPreferences 数据能被其他应用程序读，但不能写。
- Context.MODE_WORLD_WRITEABLE：指定 SharedPreferences 数据能被其他应用程序读、写。

上述的 4 个参数代表 SharedPreferences 的 4 种读写模式。在 Android 开发中，使用后两者易于出现各种安全问题，因此这两种慢慢地被官方遗弃了，现在只推荐使用前两种。当然这里说的遗弃并不是无法使用，而是不再推荐使用，如果开发者有十足的把握保证使用后两种模式不会造成安全隐患，也可以使用。

正如前文中说的，SharedPreferences 对象本身只能获取数据而不支持存储和修改，存储、修改的主要操作都是依靠 SharedPreferences.edit()获取的内部接口 Editor 对象来完成的。在 SharedPreferences 的使用过程中，慢慢总结出 SharedPreferences.Editor 中 4 种常用的方法，如表 7-1 所示。

表 7-1 SharedPreferences.Editor 中常用的方法

方法名	说明
putXxx(String key , xxx value)	向 SharedPreferences 存入指定 key 对应的数据，其中 xxx 可以是 boolean、float、int、String 等各种数据类型
remove()	删除 SharedPreferences 中指定 key 对应的数据项
commit()	当 Editor 编辑完成后，使用该方法提交修改
clear()	清空 SharedPreferences 里的所有数据

除了 SharedPreferences.Editor 中的这些存储以及修改操作方法外，在 SharedPreferences 类自身包括了一个读取数据的方法，即 getXxx(String key , xxx value)方法。它的作用就是从

SharedPreferences 中取出指定 key 对应的数据，其中 xxx 可以是 boolean、float、int、String 等各种数据类型。

下面通过一个实例来加深读者的理解。假设现在我们需要开发一款电商类的 Android 应用，当用户第一次填写了送货地址完成购物之后，第二次进入填写地址界面时直接使用保存好的地址。地址这样的信息相对较小，我们一般使用 SharedPreferences 来保存。

为了实现这样一个应用，首先需要开发一个应用界面。在这个界面中，我们使用两个 EditText 来输入姓名与地址，使用 TextView 来提示信息，并使用 Button 来触发保存事件，代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".SharedActivity">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="60dp"
        android:layout_gravity="center_vertical"
        android:orientation="horizontal">

        <TextView
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="收货人姓名："
            android:textSize="20sp" />

        <EditText
            android:id="@+id/user_name"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_weight="2" />
    </LinearLayout>

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="60dp"
        android:layout_gravity="center_vertical"
        android:orientation="horizontal">
```

```
            <TextView
                android:layout_width="0dp"
                android:layout_height="wrap_content"
                android:layout_weight="1"
                android:text="收货人地址："
                android:textSize="20sp" />

            <EditText
                android:id="@+id/user_address"
                android:layout_width="0dp"
                android:layout_height="wrap_content"
                android:layout_weight="2" />
        </LinearLayout>

        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="60dp"
            android:layout_gravity="center_vertical"
            android:orientation="horizontal">

            <Button
                android:id="@+id/save_shared"
                android:layout_width="0dp"
                android:layout_height="wrap_content"
                android:layout_weight="1"
                android:text="保存" />

            <Button
                android:id="@+id/del_shared"
                android:layout_width="0dp"
                android:layout_height="wrap_content"
                android:layout_weight="1"
                android:text="删除" />
        </LinearLayout>

    </LinearLayout>
```

为了实现存储的功能，需要在 Activity 中实现 button 的点击事件，同时为了展示读取的功能，当 SharedPreferences 中已经有数据之后，在第一次进入界面时要填充到 EditText 中，代码如下：

```
    package com.buaa.data;

    import android.content.SharedPreferences;
```

```
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;

public class SharedActivity extends AppCompatActivity implements View.OnClickListener {

    private EditText nameEditText;
    private EditText addressEditText;
    private Button savedButton;
    private Button deldButton;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_shared);
        initView();
        initData();
    }

    private void initData() {
        //创建一个 SharedPreferences 接口对象
        SharedPreferences read = getSharedPreferences("user", MODE_PRIVATE);
        //获取文件中的值
        String name = read.getString("name", "");
        String address = read.getString("address", "");
        if (name.equals("") && address.equals("")) {
            Toast.makeText(this, "抱歉，SharedPreferences 没有数据", Toast.LENGTH_LONG).show();
        } else {
            nameEditText.setText(name);
            addressEditText.setText(address);
            Toast.makeText(this, "您使用了 SharedPreferences 初始化数据",
Toast.LENGTH_LONG).show();
        }
    }

    private void initView() {
        nameEditText = (EditText) findViewById(R.id.user_name);
        addressEditText = (EditText) findViewById(R.id.user_address);
        savedButton = (Button) findViewById(R.id.save_shared);
        deldButton = (Button) findViewById(R.id.del_shared);
```

```
            savedButton.setOnClickListener(this);
            deldButton.setOnClickListener(this);
        }

        @Override
        public void onClick(View v) {
            //创建一个 SharedPreferences.Editor 接口对象，user 表示要写入的 XML 文件名
            SharedPreferences.Editor editor = getSharedPreferences("user", MODE_PRIVATE).edit();
            switch (v.getId()) {
                case R.id.save_shared:
                    String name = nameEditText.getText().toString();
                    String address = addressEditText.getText().toString();
                    //将获取的值放入文件
                    editor.putString("name", name);
                    editor.putString("address", address);
                    //提交
                    editor.commit();
                    Toast.makeText(this, "您使用了 SharedPreferences 保存数据", Toast.LENGTH_LONG).
    show();
                    break;
                case R.id.del_shared:
                    //清除所有数据
                    editor.clear();
                    editor.commit();
                    Toast.makeText(this, "您删除了 SharedPreferences 中的所有数据",
    Toast.LENGTH_LONG).show();
                    break;
            }
        }
    }
```

在代码实例中创建了一个 initDate()方法和一个 initView()方法，以及两个 Button 按钮的监听事件处理。其中，initDate()方法用于每次开启应用时初始化数据，此处初始化主要是从 ShardPreferences 中读取数据。initView()方法用于初始化 UI 界面的一些操作并对 button 的监听事件进行注册。在实现的 OnClick 方法中，先使用 switch 根据 id 判断是哪一个控件的点击事件，然后分别执行向 ShardPreferences 保存数据和从 ShardPreferences 中删除数据的操作。代码中具体执行的 ShardPreferences 以及 ShardPreferences.Editor 方法都在前文有所叙述，此处不再分析。

运行程序，当第一次进入程序或者没有用 ShardPreferences 保存数据之前运行程序，效果都会如图 7-1 中所显示的那样，使用 Toast 做一个简单的提示，告知用户没有找到之前的配置数据。当向 EditText 中输入数据并点击“保存”按钮时，也会提示已经向 SharedPreferences 中保存了数据，效果如图 7-2 所示。





图 7-1　没有可读取的数据



图 7-2　使用 ShardPreferences 进行数据存储

退出应用。再次打开应用，上一步保存在 ShardPreferences 中的数据就会被填充到 EditText 中，效果如图 7-3 所示。此时如果点击“删除”按钮，就会清空名为“user”的 ShardPreferences 中的所有数据。如果这时再次退出应用，并重新打开应用，就不会再有数据被填充到 EditText 中了，具体的效果如图 7-4 所示。



图 7-3　读取 ShardPreferences 中的数据并显示到输入框中

图 7-4　删除 ShardPreferences 中的数据

ShardPreferences 在开发中是经常被使用的技术，本节通过实例讲解了如何用 ShardPreferences 来存储数据、如何读取 ShardPreferences 中的数据以及清空 ShardPreferences 中的数据，但是这并不是全部，希望读者在学习本节内容的同时能够去阅读 Android 开发文档，以达到更深的理解。另外，本节中并没有使用 remove()方法做任何操作，读者在学习时可以试着使用它来做一些联系。

另外，使用 ShardPreferences 时，设置 setSharedPreferences(String name, int mode)方法的第二个参数是可以让应用读取外部应用数据的，也可以存储数据让外部读取，只是官方并不推荐这样做，这里就不再用实例进行讲解了。如果读者有兴趣可以尝试去实现它。

7.2　文件存储

利用 ShardPreferences 来保存数据固然有其简单轻便的优势，但是当数据较大时使用 ShardPreferences 就合适了，此时一般选择使用文件存储。

7.2.1　在应用私有文件夹中读写数据

在介绍如何在 Android 平台下进行文件存储之前有必要了解 Android 平台下的数据存储规则。在其他的操作系统如 Windows 平台下，应用程序可以自由地或者在特定的访问权限基础上访问或修改其他应用程序名下的文件等资源，而在 Android 平台下，一个应用程序中所有的数据都是私有的。

当应用程序被安装到系统中后，其所在的包会有一个文件夹用于存放自己的数据，只有这个应用程序才有对这个文件夹的写入权限，这个私有的文件夹位于 Android 系统的“\data\data\<包名>\files\”目录下，其他的应用程序都无法在这个文件夹中写入数据。除了存放私有的数据文件夹外，应用程序也具有 SDCard 卡的写入权限。

使用文件 I/O 方法可以直接往手机中存储数据，默认情况下这些文件不可以被其他应用程序访问。Android 平台支持 Java 平台下的文件 I/O 操作，主要使用 FileInputStream 和 FileOutputStream 这两个类来实现文件的存储与读取。获取这两个类对象的方式有两种。

- 第一种方式就是像 Java 平台下的实现方式一样通过构造器直接创建，如果需要向打开的文件末尾写入数据，可以通过使用构造器 FileOutputStream(File file, boolean append)将 append 设置为 true 来实现。
- 第二种获取 FileInputStream 和 FileOutputStream 对象的方式是调用 Context.openFileInput(String filename)和 Context.openFileOutput(String name,int mode)方法来创建。

在实际的 Android 开发实践中，使用第二种方式的并不多，在以往的 Android 开发类书籍里也很少有人提及这种方式。这是因为在前几年的环境下 Android 手机的机身内存还是较小的，需要借助于 SDCard，多数的文件读写操作都是针对 SDCard 的，而它只能读写位于“\data\data\<包名>\files\”下的文件，所以这种文件操作方式肯定是不合适的。但是，发展到今天，Android 手机的机身内存已足够大，不需要外加 SDCard，甚至很多手机都已经不支持 SDCard 了。智能手机的发展使得这种操作文件读写的方式变得实用了。

相信读者在 Java SE 的学习中已经熟悉了第一种方式，所以这里主要以第二种方式来进行实例讲解。其中，Context 对象提供的几个主要用于对文件操作的方法如表 7-2 所示。

表 7-2　Context 对象提供的用于文件操作的方法

方法名	说　明
openFileInput(String filename)	打开应用程序私有目录下的指定私有文件以读入数据，返回一个 FileInputStream 对象
openFileOutput(String name,int mode)	打开应用程序私有目录下的指定私有文件以写入数据，返回一个 FileOutputStream 对象，如果文件不存在就创建这个文件

（续表）

方法名	说　　明
fileList()	搜索应用程序私有文件夹下的私有文件，返回所有文件名的 String 数组
deleteFile(String fileName)	删除指定文件名的文件，成功返回 true，失败返回 false

在上述方法中，openFileOutput(String name,int mode)的第二个参数指的是读取权限，这里的权限参数和 SharedPreferences 的参数一样，有如下 4 种：

- Context.MODE_PRIVATE：指定 SharedPreferences 数据只能被本应用程序读、写。
- Context.MODE_APPEND：检查文件是否存在，存在就向文件中追加内容，否则创建新文件。
- Context.MODE_WORLD_READABLE：指定 SharedPreferences 数据能被其他应用程序读，但不能写。
- Context.MODE_WORLD_WRITEABLE：指定 SharedPreferences 数据能被其他应用程序读、写。

下面用实例来说明 Android 平台下的文件 I/O 操作方式，主要实现的功能很简单，就是在应用程序私有的数据文件夹下创建一个文件并读取其中的数据显示到屏幕的 TextView 中。在实例中，包括一个 Activity 类和对应的布局文件，以及一个文件读写类。在 Activity 中操作该文件读写类，先将文本写入文件，再从文件中读出来并展示到 TextView 中。

布局文件代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.buaa.data.activity.FileActivity">
    <TextView
        android:id="@+id/file"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="24sp" />
</RelativeLayout>
```

Activity 代码如下：

```
public class FileActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_file);

        String message = "大家好，这是清华大学出版社出的一本 Android 类书籍。";
```

```
            handFile handFile = new HandFile(this);
            handFile.writeFileData("first", message);
            String result = handFile.readFileData("first");

            TextView textView = (TextView) findViewById(R.id.file);
            textView.setText(result);
        }
}
```

文件读写类代码如下：

```
package com.buaa.data.file;

import android.content.Context;

import java.io.FileInputStream;
import java.io.FileOutputStream;

public class HandFile {

        private Context context;

        public HandFile(Context context) {
                this.context = context;
        }

        public void writeFileData(String filename, String message) {
                try {
                        FileOutputStream fileOutputStream = context.openFileOutput(filename,
context.MODE_PRIVATE);
                        byte[] bytes = message.getBytes();
                        fileOutputStream.write(bytes);
                        fileOutputStream.close();
                } catch (Exception e) {
                        e.printStackTrace();
                }
        }

        public String readFileData(String fileName) {
                String result = "";
                try {
                        FileInputStream fileInputStream = context.openFileInput(fileName);
                        int length = fileInputStream.available();
                        byte[] buffer = new byte[length];
```

```
                fileInputStream.read(buffer);
                result = new String(buffer);
                fileInputStream.close();
            } catch (Exception e) {
                e.printStackTrace();
            }
            return result;
        }
    }
```

实例代码中的新方法和逻辑以及实现的功能在前文都已经叙述了。运行程序，效果如图 7-5 所示。

data
大家好，这是清华大学出版社出的一本Android类书籍。

图 7-5　在应用私有文件夹中读写数据

7.2.2　向 SDCard 写入数据

上述的实例是向应用的私有文件夹写入文件，虽然现在向 SDCard 中写入文件已经很少了，但是下面还是要简述一下如何向 SDCard 中写入文件。

想要向 SDCard 中写入数据时必须使用上面所说的第一种方法去获得 FileInputStream 和 FileOutputStream 对象，而使用第一种方式获取 FileInputStream 和 FileOutputStream 对象必然需要传入文件路径或者该文件路径的 File 对象，并且要在获取 SDCard 路径之前判断它是否存在。以获取 FileInputStream 为例，在程序的代码如下：

```
FileInputStream fileInputStream;
if (Environment.getExternalStorageState().equals(Environment.MEDIA_MOUNTED)) {
    try {
        fileInputStream = new FileInputStream(
                Environment.getExternalStorageDirectory().toString()+
                        File.separator+yourFileName);
    } catch (FileNotFoundException e) {
        Log.e("Exception", e.toString());
    }
}
```

当然，读取 SDCard 是需要添加权限的。Android 权限是一种安全机制。Android 是基于 Linux 的，因此具有和 Linux 一样的权限管理问题。Android 权限主要用于限制应用程序内部某些具有限制性特性的功能使用以及应用程序之间的组件访问。读取 SDCard 并不是应用自身的操作，因此也需要添加权限。添加权限的方式是在 AndroidManifest.xml 文件中加入 <uses-permission android:name="****** 权限名 ****"/>，具体到本例中，应该在 AndroidManifest.xml 文件中加入下面两条权限：

```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.MOUNT_UNMOUNT_FILESYSTEMS"/>
```

很多初学者在使用权限时会将权限配置的位置放错，为了避免此问题，特将此时的 AndroidManifest.xml 文件展示如下：

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.buaa.data">

    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
    <uses-permission android:name="android.permission.MOUNT_UNMOUNT_FILESYSTEMS"/>

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".activity.FileActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

这样添加的权限是静态权限，在 Android 6.0 版本之前，这样做是可以的，但是在 6.0 版本的 Android 系统上仅仅这样操作是不行的，还需要加入动态权限设置。至于如何加入动态权限设置，将在 7.5 节进行讲解，读者可以结合 7.5 节所讲的内容对读取 SDCard 的内容进行完善。

另外，与 ShardPreferences 相同，通过参数设置也是可以读取外部应用数据的，还可以存储数据让外部读取。只是官方并不推荐这样做，这里就不再用实例进行讲解了。如果读者有兴趣可以尝试着去实现。

7.3 SQLite 数据库

每个应用程序都要使用数据，Android 应用程序也不例外。Android 使用开源的、与操作系统无关的 SQL 数据库——SQLite。

7.3.1 SQLite 简介

SQLite 数据库是 D.Richard Hipp 用 C 语言编写的开源嵌入式数据库，支持的数据库大小为 2TB。它的第一个 Alpha 版本诞生于 2000 年 5 月，是一款轻量级数据库，设计目标是嵌入式的，占用资源非常低，只需要几百千字节的内存就够了。SQLite 已经被多种软件和产品使用，Mozilla FireFox 就是使用 SQLite 来存储配置数据的，Android 和 iPhone 也是使用 SQLite 来存储数据的。SQLite 具有如下特征。

1. 轻量级

SQLite 和 C\S 模式的数据库软件不同，它是进程内的数据库引擎，因此不存在数据库的客户端和服务器。使用 SQLite 一般只需要带上一个动态库就可以享受全部功能，而且动态库的尺寸相当小。

2. 独立性

SQLite 数据库的核心引擎本身不依赖第三方软件，使用时也不需要“安装”，能够省去不少麻烦。

3. 隔离性

SQLite 数据库中的所有信息（比如表、视图、触发器）都包含在一个文件内，方便管理和维护。

4. 跨平台

SQLite 数据库支持大部分操作系统，除了在电脑上使用的操作系统之外，很多手机操作系统同样可以运行，比如 Android、Windows Mobile、Symbian、Palm 等。

5. 多语言接口

SQLite 数据库支持很多语言编程接口，比如 C\C++、Java、Python、dotNet、Ruby、Perl 等，得到更多开发者的喜爱。

6. 安全性

SQLite 数据库通过数据库级上的独占性和共享锁来实现独立事务处理。这意味着多个进程可以在同一时间从同一数据库读取数据，但只有一个可以写入数据。在某个进程或线程向数据库执行写操作之前必须获得独占锁定。在发出独占锁定后，其他的读或写操作将不会再发生。

SQLite 和其他数据库最大的不同就是对数据类型的支持，创建一个表时可以在 CREATE TABLE 语句中指定某列的数据类型，但是你可以把任何数据类型放入任何列中。当将某个值插入数据库时，SQLite 将检查数据的类型。如果该类型与关联的列不匹配，那么 SQLite 会尝试将该值转换成该列的类型。如果不能转换，那么该值将作为其本身具有的类型存储。比如可以把一个字符串（String）放入 INTEGER 列。SQLite 称之为“弱类型”。

7.3.2 SQLite 操作的核心类 SQLiteDatabase 与 SQLiteOpenHelper

在 Android 中，操作 SQLite 主要依靠 SQLiteDatabase 与 SQLiteOpenHelper 这两个类。其中，SQLiteDatabase 是用于执行数据库操作的类，SQLiteOpenHelper 是 SQLiteDatabase 的一个帮助类，用来管理数据库的创建和版本的更新。由于 SQLiteDatabase 对象是通过 SQLiteOpenHelper 调用方法来获得的，因此下面先讲解 SQLiteOpenHelper 再讲解 SQLiteDatabase。

1. SQLiteOpenHelper

为什么需要 SQLiteOpenHelper？考虑这样一种需求：在编写数据库应用软件时，开发的软件可能会安装在很多用户的手机上，如果应用使用到了 SQLite 数据库，就必须在用户初次使用软件时创建出应用使用到的数据库表结构并添加一些初始化记录，另外在软件升级的时候也需要对数据表结构进行更新。那么，如何才能实现在用户初次使用或升级软件时自动在用户的手机上创建出应用需要的数据库表呢？总不能让我们在每个需要安装此软件的手机上通过手动方式创建数据库表吧？因为这种需求是每个数据库应用都要面临的，所以 Android 系统提供了 SQLiteOpenHelper 的抽象类，通过继承它对数据库版本进行管理来实现前面提出的需求。

一个类在继承 SQLiteOpenHelper 时一般需要实现 onCreate 和 onUpgrade 方法。SQLiteOpenHelper 的主要方法如表 7-3 所示。

表7-3　SQLiteOpenHelper类的主要方法

方法名	方法描述
SQLiteOpenHelper(Context context,String name,SQLiteDatabase.CursorFactory factory, int version)	构造方法，一般是传递一个要创建的数据库名称的参数
onCreate(SQLiteDatabase db)	创建数据库时调用
onUpgrade(SQLiteDatabase db,int oldVersion , int newVersion)	版本更新时调用
getReadableDatabase()	创建或打开一个只读数据库
getWritableDatabase()	创建或打开一个读写数据库

为了实现对数据库版本进行管理，SQLiteOpenHelper 类提供了两个重要的方法，分别是 onCreate(SQLiteDatabase db)和 onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion)，前者用于初次使用软件时生成数据库表，后者用于升级软件时更新数据库表结构。

当调用 SQLiteOpenHelper 的 getWritableDatabase()或者 getReadableDatabase()方法获取用于操作数据库的 SQLiteDatabase 实例时，如果数据库不存在，Android 系统将会自动生成一个数据库，接着调用 onCreate()方法。onCreate()方法在初次生成数据库时才会被调用，在 onCreate()方法里可以生成数据库表结构并添加一些应用使用到的初始化数据。

onUpgrade()方法在数据库的版本发生变化时会被调用，一般在软件升级时才需改变版本号，而数据库的版本是由程序员控制的，假设数据库现在的版本是 1，由于业务的变更，修改了数据库表结构，这时就需要升级软件，升级软件时希望更新用户手机里的数据库表结构，为了实现这一目的，可以把原来的数据库版本设置为 2（可以随意设置，大于 1 即可），并在 onUpgrade()方法里面实现表结构的更新。当软件的版本升级次数比较多时，在 onUpgrade()方法里面可以根据原版本号和目标版本号进行判断，然后做出相应的表结构及数据更新。

使用 getWritableDatabase()和 getReadableDatabase()方法都可以获取一个用于操作数据库的 SQLiteDatabase 实例，但 getWritableDatabase() 方法以读写方式打开数据库。一旦数据库的磁盘空间满了，数据库就只能读而不能写了如果使用 getWritableDatabase()打开数据库就会出错。getReadableDatabase()方法先以读写方式打开数据库，如果数据库的磁盘空间满了就会打开失败，当打开失败后会继续尝试以只读方式打开数据库。所以在这里特别提醒读者，

getWritableDatabase()与 getReadableDatabase 的区别是，当数据库写满时调用前者会报错，调用后者则不会，所以如果不是更新数据库最好调用后者来获得数据库连接。

下面是一个典型的继承自 SQLiteOpenHelper 的类，读者可以根据上文对 SQLiteOpenHelper 的几个方法介绍来理解下面的代码。

```
public class OpenHelper extends SQLiteOpenHelper {
    private static final String name = "test.db"; //数据库名称
    private static final int version = 1; //数据库版本

    public OpenHelper(Context context) {
        super(context, name, null, version);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        //创建表
        db.execSQL("CREATE TABLE IF NOT EXISTS " +
                "user (person_id INTEGER primary key autoincrement, " +
                "name varchar(32), age INTEGER)");
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        if (newVersion > oldVersion) {
            //修改表，加一列
            db.execSQL("ALTER TABLE user ADD phone VARCHAR(11)");
        }
    }
}
```

在此特别提醒读者，在 onUpgrade()方法中要做的是更新表结构等一系列操作，请注意是更新而不是删除后重新创建。在有的 Android 书籍中给出的实例是教读者先删除原表，再创建新表，这是极端错误的，必须予以警示。试想，在原表中存有大量的用户数据，而该程序将原表删除了，这将会造成多么大的灾难！希望读者能够谨记此点。

2. SQLiteDatabase

Android 提供了一个名为 SQLiteDatabase 的类，它封装了一些操作数据库的 API，使用该类可以完成对数据进行添加（Create）、查询（Retrieve）、更新（Update）和删除（Delete）操作。

获取 SQLiteDatabase 类的对象要通过 SQLiteOpenHelper 调用 getReadableDatabase()方法，具体如下：

```
OpenHelper openHelper = new OpenHelper(context);
SQLiteDatabase db = openHelper.getReadableDatabase();
```

这里的 context 是 Context 类的对象。当获取了 SQLiteDatabase 类的对象之后就可以用该对象去对数据库进行操作了。官方也为我们提供了很多种操作方法，比较常用的如表 7-4 所示。

表 7-4 SQLiteDatabase 类的主要方法

（返回值）方法名	方法描述
(int) delete(String table,String whereClause,String[] whereArgs)	删除数据行的方法
(long) insert(String table,String nullColumnHack,ContentValues values)	添加数据行的方法
(int) update(String table, ContentValues values, String whereClause, String[] whereArgs)	更新数据行的方法
(void) execSQL(String sql)	执行一个 SQL 语句
(void) close()	关闭数据库
(Cursor) query(String table, String[] columns, String selection, String[] args, String groupBy, String having, String orderBy, String limit)	查询指定的数据表返回一个带游标的数据集
(Cursor) rawQuery(String sql, String[] args)	运行一个预置的 SQL 语句，返回带游标的数据集，与 execSQL(String sql)相比，它可以防止 SQL 注入

从这几个实例中我们发现增、删、改、查的操作都可以通过两种方式来实现；第一种是通过手动编写 SQL 语句、调用 execSQL(String sql)以及 rawQuery(String sql, String[] selectionArgs)来实现；另外一种就是通过调用系统的 insert()、delete()、update()和 query()等 api 来实现。官方提供的这些 api 是通过开发者传入的参数进行 SQL 的组装，而 Google 公司这些工程师根据传入的参数写出的 SQL 在执行效率等各个方面都是相对高效的，并且在 SQL 语句的格式上也是统一的，使用这些 api 可能甚至比一些精通 SQL 的开发者使用第一种方式要好。除此之外，api 的统一性很好，在代码的可读性以及维护性上的优势是不可小视的。因此，这里建议使用官方提供的 api 进行数据库的操作，具体如下。

（1）增加数据

insert（String table,String nullColumnHack,ContentValues values）方法用于添加数据，各个字段的数据使用 ContentValues 进行存放。ContentValues 类似于 MAP。相对于 MAP，ContentValues 提供了存取数据对应的 put(String key, Xxx value)和 getAsXxx(String key)方法，key 为字段名称，value 为字段值，Xxx 指的是各种常用的数据类型，如 String、Integer 等。使用范例如下：

```
ContentValues contentValues = new ContentValues();
contentValues .put("name","李瑞奇");
contentValues .put("age",26);
db.insert("user",null,contentValues );
db.close();
```

在上述范例代码中，不管 insert()的第三个参数是否包含数据，执行 insert()方法必然会添

加一条记录，如果第三个参数为空，就会添加一条除主键之外其他字段值为 null 的记录。insert()方法内部实际上通过构造 insert SQL 语句完成了数据的添加。

不管第三个参数是否包含数据，执行 Insert()方法必然会添加一条记录，如果第三个参数为空，会添加一条除主键之外其他字段值为 null 的记录。Insert()方法内部实际上通过构造 insert SQL 语句完成数据的添加，Insert()方法的第二个参数用于指定空值字段的名称，相信大家对该参数会感到疑惑，该参数的作用是：如果第三个参数 values 为 null 或者元素个数为 0，由于 Insert()方法要求必须添加一条除了主键之外其他字段为 null 值的记录，为了满足 SQL 语法的需要，insert 语句必须给定一个字段名，如 insert into person(name) values(null)，倘若不给定字段名，insert 语句变为 insert into person() values()，显然不满足标准 SQL 的语法。对于字段名，建议使用主键之外的字段，如果使用了 INTEGER 类型的主键字段，执行类似 insert into person(personid) values(null)的 insert 语句后，该主键字段值也不会为 null。如果第三个参数 values 不为 null 并且元素的个数大于 0，就把第二个参数设置为 null。

另外，insert()是有返回值的：当执行失败时会返回-1，其他时候返回新添加记录的行号。开发时可以据此判断是否添加成功。

（2）删除数据

delete(String table,String whereClause,String[] whereArgs)方法共有 3 个参数，第一个参数表示的是要执行操作的表，第二个参数用来过滤不需要的值或者选择适当的要素，第三个参数用于给第二个参数的占位符提供数据，其中第二个参数可以有多个条件。具体的范例如下：

```
db. delete("user", "id=？ ", new String[]{"12"});
db.close();
```

范例的意义是删除 user 表中 id=12 的数据。delete()方法也是有返回值的，它的返回值指的是删除操作影响的行数，如果返回值为 0 就意味着操作失败。开发时可以据此判断是否删除成功。

（3）更新操作

update(String table, ContentValues values, String whereClause, String[] whereArgs)方法共有 4 个参数，第一个参数表示的是要执行操作的表，第二个参数使用一个 ContentValues 对象封装要更新的列和对应的值，第三个参数用来过滤不需要的值或者选择适当的要素，第四个参数用于给第二个参数的占位符提供数据，其中第三个参数可以有多个条件。具体的范例如下：

```
ContentValues contentValues = new ContentValues();
contentValues .put("name","李瑞奇");
contentValues.put("age", 26);
db.update("user",contentValues,"id=?",new String[]{"12"});
```

范例的意义是将 user 表中 id=12 的数据的 name 修改为“李瑞奇”，age 修改为 26。Update()方法也是有返回值的，它的返回值指的是更新操作影响的行数，如果返回值为 0 就意味着操作失败。开发时可以据此判断是否更新成功。

（4）查询操作

由于查询操作的返回值是 Cursor 类的对象，因此在介绍查询操作之前要介绍 Cursor 类。在 Android 系统中，数据库查询结果的返回值并不是数据集合的完整复制，而是返回数据集的指针，这个指针就是 Cursor 类。Cursor 类支持在查询的数据集合中的多种移动方式，并能够获取数据集合的属性名称和序号。针对 Cursor，系统提供了一些操作方法，如表 7-5 所示。

表 7-5 Cursor 类的主要方法

方法名称	方法描述
getCount()	总记录条数
isFirst()	判断是否为第一条记录
isLast()	判断是否为最后一条记录
moveToFirst()	移动到第一条记录
moveToLast()	移动到最后一条记录
move(int offset)	移动到指定的记录
moveToNext()	移动到下一条记录
moveToPrevious()	移动到上一条记录
getColumnIndex(String columnName)	获得指定列索引的 int 类型值

查询操作相对前面几种操作要复杂一些，因为查询会带有很多条件。查询操作的方法 query(String table, String[] columns, String selection, String[] selectionArgs,String groupBy, String having, String orderBy, String limit)共有 8 个参数。这些参数的意义如下：

- table: 表名称，不可为空。
- colums: 想要查询的字段名称数组，可以为 null，如果为 null 就返回所有字段。
- selection: 条件子句，相当于 SQL 语句中的 where 部分，可以为空，为空时则查询所有数据。
- selectionArgs: 条件语句的参数数组，用来填充到条件子句的占位符中，当然也可以为空。
- groupBy: 分组语句，可以为空。
- having: 分组条件，可以为空。
- orderBy: 用来排序的语句，可以为空。
- limit: 用来做分页查询的限制条件，可以为空。

读者可以发现 query()方法的 8 个参数其实对应着 SQL 语句的各个部分，也和 SQL 语句一样，除了表名不可为空外，都可以为空。

查询所有数据的范例如下：

```
public List<User> queryAll() {
    List<User> userList = new ArrayList<>();
    Cursor cursor = db.query("user", null, null,
            null, null, null, null);
    while (cursor.moveToNext()) {
        User user = new User();
```

```
                user.setUserId(cursor.getInt(0));
                user.setName(cursor.getString(1));
                user.setAge(cursor.getInt(2));
                userList.add(user);
            }
            return userList;
        }
```

查询指定数据的范例如下：

```
        public User queryOne(User user) {
            Cursor cursor = db.query("user", null, "name=?",
                        new String[]{user.getName()}, null, null, null);
            while (cursor.moveToNext()) {
                user.setUserId(cursor.getInt(0));
                user.setAge(cursor.getInt(2));
            }
            return user;
        }
```

7.3.3 SQLite 操作实例

下面通过一个完整的实例来展示如何进行数据库操作。由于实例中使用的方法在前文都已经讲述过，因此不再对具体的方法进行分析，主要通过完整的实例让读者有一个宏观的认知。实例将通过创建 100 条模拟数据，用 ListView 展示出来，同时在 ListView 中对数据进行删除和更新操作。最后演示如何进行数据库版本更新的操作。

在进行具体的数据库操作之前，先建立一个 Java 实体类 User 类，再通过这个实体类进行数据的封装。User 类中包括 userId、name、age 三个属性，具体代码如下：

```
package com.buaa.data.bean;

public class User {
    private int userId;
    private String name;
    private int age;

    public int getUserId() {
        return userId;
    }

    public void setUserId(int userId) {
        this.userId = userId;
    }

    public String getName() {
```

```
            return name;
        }

        public void setName(String name) {
            this.name = name;
        }

        public int getAge() {
            return age;
        }

        public void setAge(int age) {
            this.age = age;
        }
    }
```

完成 Java 实体类的创建之后，通过继承 SQLiteOpenHelper 来实现数据库辅助类，并实现其中的 onCreate()方法和 onUpgrade()方法。在 onCreate()方法中创建表，并使表中的字段与 User 类的属性相一致。同时实现 onUpgrade()方法，以更新数据库。

```
    package com.buaa.data.dao;

    import android.content.Context;
    import android.database.sqlite.SQLiteDatabase;
    import android.database.sqlite.SQLiteOpenHelper;

    public class OpenHelper extends SQLiteOpenHelper {

        private static final String name = "test.db"; //数据库名称
        private static final int version = 1; //数据库版本

        public OpenHelper(Context context) {
            super(context, name, null, version);
        }

        @Override
        public void onCreate(SQLiteDatabase db) {
            //创建表
            db.execSQL("CREATE TABLE IF NOT EXISTS " +
                    "user (person_id INTEGER primary key autoincrement, " +
                    "name varchar(32), age INTEGER)");
        }

        @Override
```

```
        public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
            if (newVersion > oldVersion) {
                //修改表，加一列
                db.execSQL("ALTER TABLE user ADD phone VARCHAR(11)");
            }
        }
    }
```

接下来创建 UserDao 类，并在该类中实现对 User 的增、删、改、查操作，代码如下：

```
package com.buaa.data.dao;

import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;

import com.buaa.data.bean.User;

import java.util.ArrayList;
import java.util.List;

public class UserDao {
    private SQLiteDatabase db;

    public UserDao(SQLiteDatabase sqLiteDatabase) {
        this.db = sqLiteDatabase;
    }

    public boolean insert(User user) {
        ContentValues contentValues = new ContentValues();
        contentValues.put("name", user.getName());
        contentValues.put("age", user.getAge());
        long insertResult = db.insert("user", null, contentValues);
        if (insertResult == -1) {
            return false;
        }
        return true;
    }

    public boolean delete(User user) {
        int deleteResult = db.delete("user", "user_id=?  ", new String[]{user.getUserId() + ""});
        if (deleteResult == 0) {
            return false;
```

```
        }
        return true;
    }

    public boolean update(User user) {
        ContentValues contentValues = new ContentValues();
        contentValues.put("name", user.getName());
        contentValues.put("age", user.getAge());
        int updateResult = db.update("user", contentValues, "user_id=?", new String[]{user.getUserId() +
""});
        if (updateResult == 0) {
            return false;
        }
        return true;
    }

    public User queryOne(User user) {
        Cursor cursor = db.query("user", null, "name=?",
                new String[]{user.getName()}, null, null, null);
        while (cursor.moveToNext()) {
            user.setUserId(cursor.getInt(0));
            user.setAge(cursor.getInt(2));
        }
        return user;
    }

    public List<User> queryAll() {
        List<User> userList = new ArrayList<>();
        Cursor cursor = db.query("user", null, null,
                null, null, null, null);
        while (cursor.moveToNext()) {
            User user = new User();
            user.setUserId(cursor.getInt(0));
            user.setName(cursor.getString(1));
            user.setAge(cursor.getInt(2));
            userList.add(user);
        }
        return userList;
    }
}
```

当完成上述 3 个类之后，接下来就可以使用 Activity 进行各项操作了。

（1）创建数据

创建一个 Activity，并在 Activity 中创建一个循环生成 100 条数据的方法，代码如下：

```
package com.buaa.data.activity;

import android.database.sqlite.SQLiteDatabase;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;

import com.buaa.data.R;
import com.buaa.data.bean.User;
import com.buaa.data.dao.OpenHelper;
import com.buaa.data.dao.UserDao;

public class UserActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_user);
        createData();
    }

    public void createData() {
        OpenHelper openHelper = new OpenHelper(this);
        SQLiteDatabase sqLiteDatabase = openHelper.getReadableDatabase();
        UserDao userDao = new UserDao(sqLiteDatabase);
        User user = new User();
        for (int i = 0; i < 100; i++) {
            user.setName("李瑞奇" + i);
            user.setAge(26);
            userDao.insert(user);
        }
        sqLiteDatabase.close();
    }
}
```

此时运行程序，数据就创建成功了。我们将在下一部分进行验证。

（2）展示数据

在上一部分创建了 100 条数据的基础上，用 ListView 来展现这些数据。这需要修改 Activity 的布局文件，并创建一个 item 布局文件，同时在 Activity 中查询数据。由于数据中只有 name 和 age 两个文本，因此选择 SimpleAdapter 作为 ListView 的适配器。

在布局文件中加入一个 ListView 控件：

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.buaa.data.activity.UserActivity">

    <ListView
        android:id="@+id/user_list"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"></ListView>
</RelativeLayout>
```

为了能够展示出数据中的 name 和 age 两个字段，我们创建一个 item：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="50dp"
    android:orientation="horizontal">

    <TextView
        android:id="@+id/user_name"
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:layout_weight="1"
        android:gravity="center"
        android:textSize="24sp" />

    <TextView
        android:id="@+id/user_age"
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:layout_weight="1"
        android:gravity="center_vertical"
        android:textSize="24sp" />
</LinearLayout>
```

完成了这几个步骤之后，就可以在 Activity 中查询数据库并使用适配器展示到 ListView 中了。此时不再需要创建数据，不再使用之前创建数据的方法。代码如下：

```
public class UserActivity extends AppCompatActivity {
```

```
    private ListView listView;
    private List<Map<String, Object>> mapList;
    private OpenHelper openHelper;
    private List<User> userList;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_user);
        initData();
        initView();
    }

    private void initView() {
        listView = (ListView) findViewById(R.id.user_list);
        final SimpleAdapter simpleAdapter = new SimpleAdapter(
                this, mapList, R.layout.user_item,
                new String[]{"name", "age"}, new int[]{R.id.user_name, R.id.user_age});
        listView.setAdapter(simpleAdapter);
    }

    private void initData() {
        openHelper = new OpenHelper(this);
        SQLiteDatabase sqLiteDatabase = openHelper.getReadableDatabase();
        UserDao userDao = new UserDao(sqLiteDatabase);
        userList = userDao.queryAll();

        mapList = new ArrayList<>();
        for (User user : userList) {
            Map<String, Object> map = new HashMap<>();
            map.put("name", user.getName());
            map.put("age", user.getAge());
            mapList.add(map);
        }
        sqLiteDatabase.close();
    }
}
```

运行程序，就会发现创建数据时添加的 100 条数据显示在界面上了，效果如图 7-6 所示。



图 7-6 使用 SQLite 存储、读取数据

（3）删除数据

在实例中，对长按事件进行监听，当长按时询问是否删除数据，如果点击“是”就删除该条数据。实现此功能，Activity 类需要实现 AdapterView.OnItemLongClickListener 接口，并实现监听方法。此处为了简单，使用匿名内部类的方式来处理，只需要在 initView()方法中加入如下代码即可：

```
listView.setOnItemLongClickListener(new AdapterView.OnItemLongClickListener() {
    @Override
    public boolean onItemLongClick(AdapterView<?> parent, View view, int position, long id) {
        final int location = position;
        new AlertDialog.Builder(UserActivity.this)
                .setTitle("警告")
                .setMessage("确定删除此条数据吗？")
                .setPositiveButton("是", new DialogInterface.OnClickListener() {
                    @Override
                    public void onClick(DialogInterface dialog, int which) {
                        SQLiteDatabase sqLiteDatabase = openHelper.getReadableDatabase();
                        UserDao userDao = new UserDao(sqLiteDatabase);
                        userDao.delete(userList.get(location));
                        sqLiteDatabase.close();
                        mapList.remove(location);
                        simpleAdapter.notifyDataSetChanged();
                    }
                })
                .setNegativeButton("否", null)
                .show();
        return true;
    }
});
```

运行程序之后，长按一条数据，会出现提示是否删除数据的 Dialog，点击“是”删除一条数据，效果如图 7-7 所示。



图 7-7　使用 SQLite 删除数据

（4）*更新数据*

与删除一条数据的做法相同，这里使用点击事件来触发更新数据的操作。但点击一个条目时，应用会弹出一个包含 EditText 的 Dialog，可以在其中修改数据。与删除数据相同，也是用匿名内部类的方式来实现。代码如下：

```
listView.setOnItemClickListener(new AdapterView.OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
        final int location = position;
        TextView nameText = (TextView) view.findViewById(R.id.user_name);
        final EditText ageEdit = new EditText(UserActivity.this);
        //设置 EditText 的输入为数字
        ageEdit.setInputType(InputType.TYPE_CLASS_NUMBER);

        new AlertDialog.Builder(UserActivity.this)
                .setTitle("修改" + nameText.getText().toString() + "的年龄")
                .setView(ageEdit)
                .setPositiveButton("确定", new DialogInterface.OnClickListener() {
                    @Override
                    public void onClick(DialogInterface dialog, int which) {
                        String updateAge = ageEdit.getText().toString();

                        SQLiteDatabase sqLiteDatabase = openHelper.getReadableDatabase();
                        UserDao userDao = new UserDao(sqLiteDatabase);
                        User user = userList.get(location);
                        user.setAge(Integer.parseInt(updateAge));
                        userDao.update(user);
                        sqLiteDatabase.close();

                        Map<String, Object> updateMap = mapList.get(location);
                        updateMap.put("age", Integer.parseInt(updateAge));
                        mapList.remove(location);
```

```
                    mapList.add(location, updateMap);
                    simpleAdapter.notifyDataSetChanged();
                }
            })
            .setNegativeButton("取消", null)
            .show();
    }
});
```

这里实现了 ListView 的 item 点击事件。为了能够实现数据更新的操作，特别在此 Dialog 中设置了一个 EditText，用来获取输入数据，并且用 ageEdit.setInputType(InputType.TYPE_CLASS_NUMBER)方法设置 EditText 的输入必须为数字。同时，为了让修改完数据后能够使数据在 ListView 中马上改变、位置不因修改数据而导致变化，在点击“确定”按钮时还对集合 mapList 做了一系列操作，比如修改其中数据并添加回原来的位置。

运行程序，点击一条数据，会弹出一个提示框，在其中输入一个数字，点击“确定”按钮就会看到一条数据被更新，效果如图 7-8 所示。

（5）使用事务

事务处理是任何数据库都需要面对的问题，SQLite 数据也不例外。事务是一个针对数据库执行操作的工作单元。它以逻辑顺序完成的工作单位或序列，即可以由用户手动操作完成，也可以由某种数据库程序自动完成。



图 7-8　使用 SQLite 更新数据

事务是指一个或多个更改数据库的扩展。例如，如果我们正在创建一个记录或者更新一个记录或者从表中删除一个记录，那么我们正在该表上执行事务。重要的是要控制事务以确保数据的完整性和处理数据库错误。

实际上，您可以把许多的 SQLite 查询联合成一组，把所有这些放在一起作为事务的一部分进行执行。

读者们应该还记得本节向数据库增加 100 条数据的例子吧。其实对于 SQLite 来说，用 for

循环的方式循环调用 insert()方式，是非常低效的。因为 SQlite 插入数据的时候默认一条语句就是一个事务，有多少条数据就有多少次磁盘操作。这就意味着增加 100 条记录也就需要 100 次读写磁盘操作。而且不能保证所有数据都能成功插入。所以更好的方法是使用事务来处理。修改后的代码如下：

```
private void createData() {
    SQLiteDatabase sqLiteDatabase = openHelper.getReadableDatabase();
    UserDao userDao = new UserDao(sqLiteDatabase);
    sqLiteDatabase.beginTransaction();
    try {
        User user = new User();
        //批量处理操作
        for (int i = 0; i < 100; i++) {
            user.setName("李瑞奇" + i);
            user.setAge(26);
            userDao.insert(user);
        }
        sqLiteDatabase.setTransactionSuccessful(); //设置事务处理成功，不设置会自动回滚不提交
    } catch (Exception e) {
        Log.e("user Transaction",e.toString());
    } finally {
        sqLiteDatabase.endTransaction(); //处理完成
        sqLiteDatabase.close();
    }
}
```

经过测试之后，不仅成功地增加了数据，并且在效率上也提高了很多。

（6）更新数据库版本

前文已经讲述过更新数据库版本的重要性了，这里就讲解如何更新数据库版本。其实，更新数据库版本很简单，只需要修改 OpenHelper 类中的 version 属性值，系统就会自动调用 onUpgrade()方法来更新数据库版本。本例中修改如下：

```
private static final int version = 2;
```

重新运行程序，这时已经更新或者说升级数据库成功了。为了测试，给 User 类加上一个 String 类型的属性 phone，在 queryAll()方法中加上一条 user.setPhone(cursor.getString(3))，然后使用 queryAll()方法，运行程序并不会报错，虽然没有查询到 phone 字段的数据，但至少说明更新数据库版本的尝试是成功的，因为数据库确实多了一列。

7.4 ContentProvider

在 Android 开发中，有时用户确实需要在应用之间进行数据的交换。通过前面几节的学习，

知道通过指定文件的操作模式为 Context.MODE_WORLD_READABLE 或 Context.MODE_WORLD_WRITEABLE 同样也可以对外共享数据。但是如果采用文件操作模式对外共享数据，数据的访问方式会因数据存储的方式而不同，导致数据的访问方式无法统一，比如采用 xml 文件对外共享数据，需要进行 xml 解析才能读取数据；采用 sharedpreferences 共享数据，需要使用 sharedpreferences API 读取数据。为此，Google 提供了 ContentProvider（内容提供者），它可以实现统一的数据访问方式。

7.4.1 ContentProvider 常用类简介

ContentProvider（内容提供者）是 Android 中的四大组件之一，主要用于对外共享数据，也就是通过 ContentProvider 把应用中的数据共享给其他应用访问，其他应用可以通过 ContentProvider 对指定应用中的数据进行操作。ContentProvider 分为系统的和自定义的，系统的例如联系人、图片等数据。内容提供者提供的数据可以存储于文件系统、SQLite 数据库或其他方式。

当应用需要通过 ContentProvider 对外共享数据时，第一步需要继承 ContentProvider 并重写表 7-6 中的方法。

表 7-6 需要被重写的 ContentProvider 类的主要方法

方法	作用
public boolean onCreate()	在创建 ContentProvider 时调用此方法
Public Cursor query(Uri uri, String[] args1, String str1, String[] args2, String str2)	用于查询指定 Uri 的 ContentProvider，返回值为一个 Cursor 类型的数据集
public Uri insert(Uri uri, ContentValues cv)	用于添加数据到指定 Uri 的 ContentProvider 中，并返回一个 Uri
public int update(Uri uri, ContentValues cv,String string, String[] stringArgs)	用于更新指定 Uri 的 ContentProvider 中的数据
public int delete(Uri uri, String string, String[] stringArgs)	用于从指定 Uri 的 ContentProvider 中删除数据
public String getType(Uri uri)	用于返回指定的 Uri 中数据的 MIME 类型

在这些方法中，getType(Uri uri)方法比较难以理解。此方法会根据传进来的 URI 生成一个代表 MimeType 的字符串，而此字符串的生成也有规则：

- 如果是单条记录，应该返回以 vnd.android.cursor.item/为首的字符串。
- 如果是多条记录，应该返回以 vnd.android.cursor.dir/为首的字符串。
- 至于字符串“/”后的字符，可以随便定义。

这里考虑一个问题，即为什么我们返回的 MimeType 要以 vnd.android.cursor.item/或 vnd.android.cursor.dir/开头。我们知道，MIME 类型其实就是一个字符串，中间用一个“/”来隔开，“/”前面是系统识别的部分，相当于我们定义一个变量时的变量数据类型，通过这个“数据类型”，系统能够知道我们所要表示的是什么内容。“/”后面的就是我们自己定义的“变量名”。

第二步需要在 AndroidManifest.xml 中对 ContentProvider 进行配置。为了能让其他应用找到该 ContentProvider，采用 authorities（主机名/域名）对 ContentProvider 进行唯一标识，可以把 ContentProvider 看作一个网站，authorities 就是它的域名。同时，为了使其他应用能够访问到这个 ContentProvider，将 android:exported 属性设置为 true。而发布的内容提供者可能会被多个应用使用，所以将 android:multiprocess 属性设置为 true。具体的做法就是将如下格式的代码放置于 application 节点下：

```
<provider
    android:name="com.buaa.data.provider.UserProvider"
    android:authorities="com.buaa.data.provider.userProdiver"
    android:exported="true"
    android:multiprocess="true" />
```

继承了 ContentProvider 类，并完成上述的方法重写以及在 AndroidManifest.xml 中的配置之后，这个应用就可以称为 ContentProvider 了。当外部应用需要对 ContentProvider 中的数据进行添加、删除、修改和查询操作时，可以使用 ContentResolver 类来完成，它提供了与 ContentProvider 相对应的增、删、改、查的方法。要获取 ContentResolver 对象，可以使用 Context 提供的 getContentResolver()方法。

通过前面对 ContentProvider 的介绍，会发现在它的几个方法中都有 Uri。通过对它的简单描述，可以指定它是跟数据相关的。准确地说，Uri 代表了要操作的数据，Uri 主要包含两个信息：需要操作的 ContentProvider 以及对 ContentProvider 中的什么数据进行操作。一个 Uri 通常由三部分组成：

（1）scheme:ContentProvider 的 scheme 已经由 Android 规定为 content://。

（2）主机名（Authority）：用于唯一标识 ContentProvider，外部调用者可以根据这个标识来找到它。

（3）路径（path）：可以用来表示我们要操作的数据，路径的构建应根据业务而定。

对于这种描述，初学者可能还是不易理解，下面举 3 个例子来帮助读者理解 Uri。例如，要操作 user 中 id 为 10 的记录，Uri 应该写成 content://com.buaa.data.provider.userProdiver/user/10；要操作 user 中 id 为 10 的记录的 name 字段，就需要在上面的 Uri 最后加上 name，即 content://com.buaa.data.provider.userProdiver/user/10/name；如果要操作 user 中的所有记录，Uri 则应该为 content://com.buaa.data.provider.userProdiver/user。

要获取一个 Uri 对象，只需调用 Uri 的静态方法 parse()，就可以按照由上文格式组成的 Uri 字符串转化为 Uri 对象了：Uri uri = Uri.parse("content://com.buaa.data.provider.userProdiver/user")。

另外，Uri 代表了要操作的数据，所以在开发中会经常需要解析 Uri，并从 Uri 中获取数据。Android 系统提供了两个用于操作 Uri 的工具类，分别为 UriMatcher 和 ContentUris。

其中，UriMatcher 类用于匹配 Uri。具体来说，UriMatcher 会在内容提供者中注册需要的 Uri，并在后面的使用中用 UriMatcher 类的 match(Uri uri)来进行匹配，用于确认是不是合法的 Uri。使用它需要进行如下 3 步操作：

步骤 01 对 UriMatcher 类进行初始化。初始化使用的代码是 UriMatcher matcher = new UriMatcher(UriMatcher.NO_MATCH)。其中，常量 UriMatcher.NO_MATCH 表示如果在之后调用

match(Uri uri)进行匹配操作，不匹配任何路径时的返回码。

步骤 02 对 Uri 进行注册。使用 UriMatcher 对象调用 addURI()就可以把需要匹配的 Uri 进行注册了。这里具体说明一下它们的意义。

执行 matcher.addURI("com.buaa.data.provider.userProdiver", "user", USER) 代码时，表示如果 match()方法匹配 content://com.buaa.data.provider.userProdiver/user 路径就匹配成功，并返回匹配码为 USER。

执行 matcher.addURI("com.buaa.data.provider.userProdiver", "user/#", USER_ID)代码时，表示如果 match()方法匹配 content://com.buaa.data.provider.userProdiver/user/20 路径就匹配成功，并返回匹配码为 USER_ID。上述代码中的“#”为通配符。

步骤 03 注册完需要匹配的 Uri 后就可以使用 matcher .match(uri)方法对输入的 Uri 进行匹配了，如果匹配就返回匹配码。

ContentUris 类用于获取 Uri 路径后面的 ID 部分，它有两个比较实用的方法，即 withAppendedId(uri, id)与 parseId(uri)。

（1）withAppendedId(uri, id)：此方法主要用于为 Uri 路径加上 id 部分。使用范例如下：

```
Uri uri = Uri.parse("content://com.buaa.data.provider.userProdiver/user")
Uri resultUri = ContentUris.withAppendedId(uri, 10);
```

经过上述操作后，生成的 Uri 为 content://com.buaa.data.provider.userProdiver/user/10。

（2）parseId(uri)：此方法主要用于从 Uri 路径中获取 ID 部分。使用范例如下：

```
Uri uri = Uri.parse("content://com.buaa.data.provider.userProdiver/user/15");
long personid = ContentUris.parseId(uri);
```

通过上述步骤就可以获取 Uri 路径的 id 部分为 15。

7.4.2　自定义 ContentProvider

在学习了 ContentProvider 的几个常用类和如何创建一个内容提供者的方法之后，下面我们用一个实例来进一步加深理解。实例先在 7.3 节内容的基础上增加一个 ContentProvider，再创建一个新的应用，在新应用中去操作这个 ContentProvder 的数据。

首先创建 UserProvider 类，并让 UserProvider 类继承 ContentProvider 类，并实现 onCreate()等方法。这些方法主要是对数据库进行操作，代码如下：

```
package com.buaa.data.provider;

import android.content.ContentProvider;
import android.content.ContentUris;
import android.content.ContentValues;
import android.content.UriMatcher;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.net.Uri;
```

```
import android.support.annotation.Nullable;

import com.buaa.data.dao.OpenHelper;

public class UserProvider extends ContentProvider {
    private OpenHelper openHelper;

    // 若不匹配采用 UriMatcher.NO_MATCH(-1)返回
    private static final UriMatcher MATCHER = new UriMatcher(UriMatcher.NO_MATCH);

    // 匹配码
    private static final int USER = 1;
    private static final int USER_ID = 2;

    static {
        // 匹配返回 CODE_NOPARAM，不匹配返回-1
        MATCHER.addURI("com.buaa.data.provider.userProdiver", "user", USER);

        // 匹配返回 CODE_PARAM，不匹配返回-1
        MATCHER.addURI("com.buaa.data.provider.userProdiver", "user/#", USER_ID);
    }

    @Override
    public boolean onCreate() {
        openHelper = new OpenHelper(this.getContext());
        return true;
    }

    /**
     * 外部应用向本应用插入数据
     */
    @Override
    public Uri insert(Uri uri, ContentValues values) {
        SQLiteDatabase db = openHelper.getWritableDatabase();
        switch (MATCHER.match(uri)) {
            case USER:
                long id = db.insert("user", "name", values);
                Uri insertUri = ContentUris.withAppendedId(uri, id);
                return insertUri;
            default:
                throw new IllegalArgumentException("this is unkown uri:" + uri);
        }
    }
```

```
/**
 * 外部应用向本应用删除数据
 */
@Override
public int delete(Uri uri, String selection, String[] selectionArgs) {
    SQLiteDatabase db = openHelper.getWritableDatabase();
    switch (MATCHER.match(uri)) {
        case USER:
            return db.delete("user", selection, selectionArgs); // 删除所有记录
        case USER_ID:
            long id = ContentUris.parseId(uri); // 取得跟在 Uri 后面的数字
            String where = "user_id = " + id;
            if (null != selection && !"".equals(selection.trim())) {
                where += " and " + selection;
            }
            return db.delete("user", where, selectionArgs);
        default:
            throw new IllegalArgumentException("this is unkown uri:" + uri);
    }
}

/**
 * 外部应用向本应用更新数据
 */
@Override
public int update(Uri uri, ContentValues values, String selection, String[] selectionArgs) {
    SQLiteDatabase db = openHelper.getWritableDatabase();
    switch (MATCHER.match(uri)) {
        case USER:
            return db.update("user", values, selection, selectionArgs); // 更新所有记录
        case USER_ID:
            long id = ContentUris.parseId(uri); // 取得跟在 Uri 后面的数字
            String where = "user_id = " + id;
            if (null != selection && !"".equals(selection.trim())) {
                where += " and " + selection;
            }
            return db.update("user", values, where, selectionArgs);
        default:
            throw new IllegalArgumentException("this is unkown uri:" + uri);
    }
}

/**
 * 如果是单条记录，应该返回以 vnd.android.cursor.item/ 为首的字符串
```

```
     * 如果是多条记录，应该返回以 vnd.android.cursor.dir/ 为首的字符串
     */
    @Override
    public String getType(Uri uri) {
        switch (MATCHER.match(uri)) {
            case USER:
                return "vnd.android.cursor.item/user";
            case USER_ID:
                return "vnd.android.cursor.item/user";
            default:
                throw new IllegalArgumentException("this is unkown uri:" + uri);
        }
    }

    @Override
    public Cursor query(Uri uri, String[] projection, String selection, String[] selectionArgs, String sortOrder)
{
        SQLiteDatabase db = openHelper.getReadableDatabase();
        switch (MATCHER.match(uri)) {
            case USER:
                return db.query("user", projection, selection, selectionArgs, null, null, sortOrder);
            case USER_ID:
                long id = ContentUris.parseId(uri);
                String where = "user_id = " + id;
                if (null != selection && !"".equals(selection.trim())) {
                    where += " and " + selection;
                }
                return db.query("user", projection, where, selectionArgs, null, null, sortOrder);
            default:
                throw new IllegalArgumentException("this is unkown uri:" + uri);
        }
    }
}
```

代码编写完成后，需要在 AndroidManifest.xml 文件中进行配置。在 Application 节点下面增加一条配置：

```
<provider
    android:name="com.buaa.data.provider.UserProvider"
    android:authorities="com.buaa.data.provider.userProdiver"
    android:exported="true"
    android:multiprocess="true" />
```

运行应用程序，应用本身并没有变化。但是，此时它已经不是一个简单的应用了，而是变成了一个内容提供者。

创建一个新的应用，在此类中使用 ContentResolver 来获取 ContentProvider 中的数据，并通过 ListView 展示出来。因为这里的界面以及 Java 实体类与 7.3 节并无区别，所以这里 Activity 的布局文件、item 的布局文件、Java 实体类都直接从 7.3 节复制过来。这个新应用中与 7.3 节应用最大的区别在于获取数据的方式不同，在这个新应用中获取数据的代码如下：

```
package com.buaa.contentresolve;

import android.content.ContentResolver;
import android.content.ContentUris;
import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
import android.net.Uri;

import java.util.ArrayList;
import java.util.List;

public class UserResolve {

    private ContentResolver resolver;
    private Uri uri;

    public UserResolve(Context context) {
        resolver = context.getContentResolver();
        uri = Uri.parse("content://com.buaa.data.provider.userProdiver/user");
    }

    public void insert(User user) {
        ContentValues values = new ContentValues();
        values.put("name", user.getName());
        values.put("age", user.getAge());
        values.put("phone", user.getPhone());
        resolver.insert(uri, values);
    }

    public void delete(User user) {
        resolver.delete(ContentUris.withAppendedId(uri, user.getUserId()), null, null);
    }

    public void update(User user) {
        ContentValues values = new ContentValues();
        values.put("age", user.getAge());
        resolver.update(ContentUris.withAppendedId(uri, user.getUserId()), values, null, null);
    }
```

```
        public List<User> query() {
            List<User> userList = new ArrayList<>();
            Cursor cursor = resolver.query(uri, null, null, null, null);
            while (cursor.moveToNext()) {
                User user = new User();
                user.setUserId(cursor.getInt(0));
                user.setName(cursor.getString(1));
                user.setAge(cursor.getInt(2));
                user.setPhone(cursor.getString(3));
                userList.add(user);
            }
            return userList;
        }
    }
```

仔细观察上述代码，就会发现这些代码与直接使用数据库获取代码十分相似。其实这是正常的，对本应用来说，另外一个应用中的数据与数据库中的数据并无太大区别。下面在 Activity 中调用 UserResolve 类中的方法来用 ListView 展示数据以及点击修改数据和长按删除数据。这部分的代码和 7.3 节的代码大体相同，只需将 7.3 节获取 UserDao 类对象的过程改成获取 UserResolve 类对象即可。

运行 ContentResolve 应用，展示数据如图 7-9 所示。运行 ContentProvider 的应用也就是 data 应用，展示数据如图 7-10 所示。可以发现两者的数据是一致的，这是因为 ContentResolve 应用中获取的数据来源于 data 应用。



图 7-9　ContentResolve 应用中的数据



图 7-10　data 应用中的数据

此时，如果点击修改数据或者长按删除一条数据，不管是在哪一个应用中进行操作，另外一个应用中的数据都会发生变化。读者可以尝试进行验证。

7.5　动态权限

在之前的章节中，我们提到了权限问题，以及新版本中的动态权限问题，但是并没有进

行详细的讲解。本节将详细介绍动态权限的由来、需要动态申请的权限有哪些，并用读取通话记录这样一个实例来让读者有个更直观的认识。

7.5.1 动态权限简介

Android 6.0 为了保护用户隐私，将一些权限的申请放在了应用运行时申请， 比如以往的开发中，开发人员只需要将需要的权限在清单文件中配置即可，安装后用户可以在设置中的应用信息中看到：XX 应用已获取****权限。用户点击可以选择给应用相应的权限。此前的应用权限用户可以选择允许、提醒和拒绝。在安装的时候用户是已经知道应用需要的权限的。但是这样存在一个问题，就是用户在安装的时候，应用需要的权限十分多（有些开发者为了省事，会请求一些不必要的权限或者请求全部的权限），这个时候用户在安装应用的时候也许并没有发现某些侵犯自己隐私的权限请求，安装之后才发现自己的隐私数据被窃取。

其实 Android 6.0 动态权限一方面是为了广大用户考虑，另一方面是 Google 为了避免一些不必要的官司，虽然这样对开发者来说可能会造成不小的困扰。不过新的权限机制中也不是所有权限都需要动态申请。在新的权限机制中，可将权限分为两类，一类是普通权限，一类是危险权限。在开发中普通权限只要使用静态权限即可，而危险权限才需要使用动态权限。官方提供了危险权限的列表，如图 7-11 所示。

Permission Group	Permissions
android.permission-group.CALENDAR	• android.permission.READ_CALENDAR • android.permission.WRITE_CALENDAR
android.permission-group.CAMERA	• android.permission.CAMERA
android.permission-group.CONTACTS	• android.permission.READ_CONTACTS • android.permission.WRITE_CONTACTS • android.permission.GET_ACCOUNTS
android.permission-group.LOCATION	• android.permission.ACCESS_FINE_LOCATION • android.permission.ACCESS_COARSE_LOCATION
android.permission-group.MICROPHONE	• android.permission.RECORD_AUDIO
android.permission-group.PHONE	• android.permission.READ_PHONE_STATE • android.permission.CALL_PHONE • android.permission.READ_CALL_LOG • android.permission.WRITE_CALL_LOG • com.android.voicemail.permission.ADD_VOICEMAIL • android.permission.USE_SIP • android.permission.PROCESS_OUTGOING_CALLS
android.permission-group.SENSORS	• android.permission.BODY_SENSORS
android.permission-group.SMS	• android.permission.SEND_SMS • android.permission.RECEIVE_SMS • android.permission.READ_SMS • android.permission.RECEIVE_WAP_PUSH • android.permission.RECEIVE_MMS • android.permission.READ_CELL_BROADCASTS
android.permission-group.STORAGE	• android.permission.READ_EXTERNAL_STORAGE • android.permission.WRITE_EXTERNAL_STORAGE

图 7-11 官方提供的危险权限的列表

Android 6.0 系统默认为 targetSdkVersion 小于 23 的应用默认授予了所申请的所有权限，针对 targetSdkVersion 大于等于 23 的，又使用了危险权限的，我们可以使用下面的几个方法来动态申请权限。

- int checkSelfPermission(String permission)：用来检测应用是否已经具有权限，这个方法是在API23 中才有的，为了兼容低版本，建议使用v4 包中的ContextCompat.checkSelfPermission (String permission)。
- void requestPermissions(String[] permissions, int requestCode)：进行请求单个或多个权限，第一个参数是请求的权限集合，第二个参数是请求码，在回调监听中可以用来判断是哪个权限请求的结果。
- void onRequestPermissionsResult(int requestCode, String[] permissions, int[] grantResults)：申请权限做出响应后的回调函数，请求成功或者失败的监听。

7.5.2 读取通话记录

在 Android 中提供了很多系统 ContextProvider，通话记录就是其中的一个典型代表。下面我们以读取通话记录为例，展示如何读取系统自带的 ContextProvider 以及动态权限的处理。和操作自定义的 ContextProvider 一样，操作系统的 ContextProvider 也是使用 ContentResolver 类。本实例中主要是读取通话记录，因此只需调用 query()方法，传入 URI 即可。

为了实现读取通话记录的功能，在 Activity 对应的布局文件 activity_call.xml 中添加了一个 ListView，代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context="com.buaa.data.activity.CallActivity">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="50dp"
        android:orientation="horizontal">

        <TextView
            android:text="号码"
            android:layout_width="0dp"
            android:layout_height="match_parent"
            android:layout_weight="1"
            android:gravity="center"
            android:textSize="26sp" />
```

```
        <TextView
            android:text="时间"
            android:layout_width="0dp"
            android:layout_height="match_parent"
            android:layout_weight="1"
            android:gravity="center"
            android:textSize="26sp" />
    </LinearLayout>

    <ListView
        android:id="@+id/call_list"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"></ListView>

</LinearLayout>
```

同时，实现一个展示条目的布局文件 call_item.xml：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="50dp"
    android:orientation="horizontal">

    <TextView
        android:id="@+id/call_mobile"
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:layout_weight="1"
        android:gravity="center"
        android:textSize="24sp" />

    <TextView
        android:id="@+id/call_date"
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:layout_weight="1"
        android:gravity="center"
        android:textSize="24sp" />
</LinearLayout>
```

在 Activity 中获取 ListView 并将从 ContentProvider 中读取的数据传入 ListView 中。在处理过程中实现动态的申请权限，代码如下：

```
package com.buaa.data.activity;

import android.Manifest;
import android.content.ContentResolver;
import android.content.pm.PackageManager;
import android.database.Cursor;
import android.os.Build;
import android.provider.CallLog;
import android.support.v4.app.ActivityCompat;
import android.support.v4.content.ContextCompat;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.util.Log;
import android.widget.ListView;
import android.widget.SimpleAdapter;
import android.widget.Toast;

import com.buaa.data.R;

import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Date;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

public class CallActivity extends AppCompatActivity {

    private ListView listCalls;
    private List<Map<String, Object>> mapList;
    //权限申请的请求码
    private static final int REQUEST_CODE = 0;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_call);
        onShowCallLog();
    }

    private void initView() {
        listCalls = (ListView) super.findViewById(R.id.call_list);
        SimpleAdapter simpleAdapter = new SimpleAdapter(this, mapList, R.layout.call_item,
```

```
                new String[]{CallLog.Calls.NUMBER, CallLog.Calls.DATE},
                new int[]{R.id.call_mobile, R.id.call_date});
        listCalls.setAdapter(simpleAdapter);
    }

    private void initData() {
        ContentResolver contentResolver = getContentResolver();
        //调用 query 方法，传入 URI 参数，即 CallLog.Calls.CONTENT_URI
        //本节希望读取电话号码与事件两个字段，传入一个包含字段名的数组
        Cursor cursor = contentResolver.query(CallLog.Calls.CONTENT_URI,
                new String[]{CallLog.Calls.NUMBER, CallLog.Calls.DATE}, null, null, null);
        mapList = new ArrayList<>();
        SimpleDateFormat dateFormater = new SimpleDateFormat("yyyy-MM-dd");
        while (cursor.moveToNext()) {
            Map<String, Object> map = new HashMap<>();
            map.put(CallLog.Calls.NUMBER, cursor.getString(0));
            map.put(CallLog.Calls.DATE, dateFormater.format(new Date(cursor.getLong(1))));
            mapList.add(map);
        }
    }

    public void onShowCallLog() {
        if (Build.VERSION.SDK_INT >= 23) {
            int checkCALL_LOGPermission = ContextCompat.
                    checkSelfPermission(this, Manifest.permission.READ_CALL_LOG);
            if (checkCALL_LOGPermission != PackageManager.PERMISSION_GRANTED) {
                //用以申请权限的方法，此时使用 ActivityCompat 类的该方法，以便于版本兼容
                ActivityCompat.requestPermissions(this,
                        new String[]{Manifest.permission.READ_CALL_LOG},
                        REQUEST_CODE);
                return;
            } else {
                //如果已经获取了相关权限，调用 initData()与 initView()方法
                initData();
                initView();
            }
        } else {
            //如果 api 版本低于 23，直接调用 initData()与 initView()方法
            initData();
            initView();
        }
    }
```

```
    //申请权限做出响应后的回调函数
    @Override
    public void onRequestPermissionsResult(int requestCode, String[] permissions, int[] grantResults) {
        switch (requestCode) {
            case REQUEST_CODE:
                if (grantResults[0] == PackageManager.PERMISSION_GRANTED) {
                    Toast.makeText(this, "获取权限成功", Toast.LENGTH_SHORT)
                            .show();
                    //获取权限成功，调用 initData()与 initView()方法
                    initData();
                    initView();
                } else {
                    Toast.makeText(this, "获取权限失败", Toast.LENGTH_SHORT)
                            .show();
                    //获取权限失败，退出 Activity
                    this.finish();
                }
                break;
            default:
                super.onRequestPermissionsResult(requestCode, permissions, grantResults);
        }
    }
}
```

同时，还需要在 AndroidMa.xml 文件中加入一条读取通话记录的用户权限：

```
<uses-permission android:name="android.permission.READ_CALL_LOG"/>
```

此时，运行应用，会提示用户是否授予读取通话记录的权限，如图 7-12 所示。如果点击“ALLOW”，应用就能够获取相关的权限、读取通话记录并展示在 ListView 中，如图 7-13 所示。



图 7-12　系统询问是否允许应用获取权限



图 7-13　读取通话记录并展示在 ListView 中

另外，在之前的章节中讲解了 Fragment，而在 Fragment 中会存在使用动态权限的状况。在 Fragment 中申请权限，一般不使用 ActivityCompat.requestPermissions()，而是直接使用 Fragment 的 requestPermissions()方法，否则会回调到 Activity 的 onRequestPermissionsResult()方法。如果在 Fragment 中嵌套 Fragment，在子 Fragment 中使用 requestPermissions()方法，onRequestPermissionsResult()不会回调回来，建议使用 getParentFragment().requestPermissions()方法，这个方法会回调到父 Fragment 中的 onRequestPermissionsResult()，加入以下代码可以把回调透传到子 Fragment：

```
@Override
public void onRequestPermissionsResult(int requestCode, String[] permissions, int[] grantResults) {
    super.onRequestPermissionsResult(requestCode, permissions, grantResults);
    List<Fragment> fragments = getChildFragmentManager().getFragments();
    if (fragments != null) {
        for (Fragment fragment : fragments) {
            if (fragment != null) {
                fragment.onRequestPermissionsResult(requestCode,permissions,grantResults);
            }
        }
    }
}
```

7.6 小　　结

数据操作是任何一门编程语言的核心，对 Android 编程来说也是如此。本章讲述了 4 种数据存储的方式，每一种都经常在实际开发中使用。对于每一种数据存储方式，读者都做到熟练操作、深刻理解内涵。另外，在 Android 6.0 版本发行之后，引入了动态权限的概念，在向应用中添加相关权限时应该加以注意。



第 8 章 Service 详解

经过之前的学习读者应该应该已经能够开发出一款包括 UI 页面并具备基本功能的应用了。但是，如果应用中需要在后台运行一些耗时操作，比如说后台播放音乐（这需要多媒体技术，在之后的章节中会讲解，此时只需关注后台播放功能），就需要使用到 Android 中的另一项技术，即 Service。Service 是 Android 系统中的四大组件之一，是一种长生命周期、没有可视化界面、运行于后台的一种服务程序。Android 四大组件中的 Activity 与 ContentProvider 都已介绍过，本章将具体讲解 Service。

8.1 Service 综述

Service（服务）是一个没有用户界面、在后台运行、执行耗时操作的应用组件。其他应用组件能够启动 Service，并且当用户切换到其他应用场景时，Service 将持续在后台运行。另外，一个组件能够绑定到一个 Service 与之交互（IPC 机制），例如，一个 Service 可能会处理网络操作、播放音乐、操作文件 I/O 或者与内容提供者（content provider）交互，所有这些活动都是在后台进行的。

8.1.1 Service 的分类

Service 可以通过运行地点和运行类型两种方式来进行分类。按运行地点分类，可以分为本地服务和远程服务两类，如表 8-1 所示。

表8-1 本地服务和远程服务

类别	区别	优点	缺点	应用
本地服务（Local）	该服务依附在主进程上	服务依附在主进程上，而不是独立的进程，在一定程度上节约了资源，另外 Local 服务因为是在同一进程，因此不需要 IPC，也不需要 AIDL。相应 bindService 会方便很多	主进程被 Kill 后，服务便会终止	常见的应用如酷我音乐播放服务
远程服务（Remote）	该服务是独立的进程	服务为独立的进程，对应进程名格式为所在包名加上所指定的 android:process 字符串。由于是独立的进程，因此在 Activity 所在进程被 Kill 的时候，该服务依然在运行，不受其他进程影响，有利于为多个进程提供服务，具有较高的灵活性	该服务是独立的进程，会占用一定资源，并且使用 AIDL 进行 IPC 稍微麻烦一点	一些提供系统服务的 Service，这种 Service 是常驻的

在实际的开发实践中，Remote Service 相对是很少见的，并且一般都是系统服务，所以本章讲解的重点是 Local Service。

按运行类型分类，Service 也可以分为两类，如表 8-2 所示。

表8-2 前台服务与后台服务

类别	区别	应用
前台服务	会在通知一栏显示 ONGOING 的 Notification	当服务被终止的时候，通知一栏的 Notification 也会消失，对于用户有一定的通知作用，常见的如音乐播放服务
后台服务	默认的服务即为后台服务，即不会在通知一栏显示 Notification	当服务被终止的时候，用户是看不到效果的，包括一些不需要运行或终止提示的服务，如天气更新、日期同步、邮件同步等

有的读者可能会问，后台服务可不可以通过自己创建的 Notification（关于 Notification 的

相关知识会在 8.4 节详细讲解，此处知道即可）来成为前台服务？答案是否定的，前台服务是在做了上述工作之后，再调用 startForeground()（Android 2.0 及其以后版本）或 setForeground()（Android 2.0 以前的版本）。这样做的好处在于，当服务被外部强制终止的时候，Notification 仍然会移除。

8.1.2 为什么不使用线程

使用服务来进行一个后台长时间的动作其实就是为了不阻塞线程，然而，Thread 就可以达到这个效果，为什么不直接使用 Thread 去代替服务呢？两者的区别如下：

（1）Thread：程序执行的最小单元，是分配 CPU 的基本单位，可以用来执行一些异步操作。

（2）Service：Android 的一种机制，当它运行的时候如果是 Local Service，那么对应的 Service 是运行在主进程的 Main 线程上的。例如，onCreate()、onStart()这些函数在被系统调用的时候都是在主进程的 Main 线程上运行的。如果是 Remote Service，那么对应的 Service 是运行在独立进程的 Main 线程上的。

通过对比可以发现两者完全不同，但是并没有说明为什么一定要使用 Service 而不是 Thread。真正的原因和 Android 的系统机制有关。Thread 的运行是独立于 Activity 的，也就是说当一个 Activity 被 finish 之后，如果没有主动停止 Thread 或者 Thread 里的 run 方法没有执行完毕，Thread 就会一直执行。因此这里会出现一个问题：当 Activity 被 finish 之后，就不再持有该 Thread 的引用。另一方面，你没有办法在不同的 Activity 中对同一 Thread 进行控制。

举个例子：如果 Thread 需要不停地隔一段时间就连接服务器做某种同步，那么该 Thread 需要在 Activity 没有 start 的时候就运行。这时 start 一个 Activity 的话就没有办法在该 Activity 里面控制之前创建的 Thread，因此需要创建并启动一个 Service，在 Service 里创建、运行并控制 Thread，这样便解决了该问题（因为任何 Activity 都可以控制同一 Service，而系统也只会创建一个对应 Service 的实例）。

8.1.3 Service 的创建与启动

Service 不能自己运行，需要通过调用 Context.startService()或 Context.bindService()方法启动服务。我们通常会把 Service 的两种启动方式称为 start 方式和 bind 方式。在表 8-3 中我们概括性地介绍了这两种方式，具体代码层面的操作会在后面通过实例的方式向读者展现。

表8-3 start方式和bind方式开启服务

开启方式	开启步骤	特点
Start 方式	① 定义一个类继承 Service ② 在 Manifest.xml 文件中配置该 Service ③ 使用 Context 的 startService()方法启动该 Service ④ 不使用时，调用 stopService()方法停止该服务	以 start 方式启动的 Service 一旦开启就和开启者没有关系了，无论开启者退出或者被销毁，Service 都依旧会运行。开启者无法调用服务中的方法

（续表）

开启方式	开启步骤	特点
Bind 方式	① 定义一个类继承 Service ② 在 Manifest.xml 文件中配置该 Service ③ 使用 Context 的 bindService(Intent intent, ServiceConnection serviceConnection, int f)方法绑定 Service ④ 不使用时调用 unbindService(ServiceConnection service-Connection)方法停止该服务。	以 bind 方式开启（绑定）的 Service，绑定者与服务绑定在了一起，绑定者一旦退出，服务就会终止，具有“不求同生，但求同死”的特点。绑定者可以调用服务中的方法。

如果只是想要启动一个后台服务长期进行某项任务，使用 start 方式就可以。

想要与正在运行的 Service 进行通信或者需要更新 Service 的状态，可以使用两种方法：一种是使用 Broadcast（第 9 章会讲解），另外一种是使用 bind 方式开启 Service。前者的缺点是如果交流较为频繁，容易造成性能上的问题，并且 BroadcastReceiver（BroadCast 中的常用类，第 9 章会进行讲解）本身执行代码的时间是很短的（也许执行到一半，后面的代码便不会执行，在开发的实践中出现过类似情况），而后者则没有这些问题，因此我们选择使用 bind 方式开启 Service。

当然，在 Android 开发实践中会经常出现两种开启方式混合使用的状态。

8.1.4 Service 生命周期

先来观察图 8-1，这是官方提供的 Service 的生命周期图。

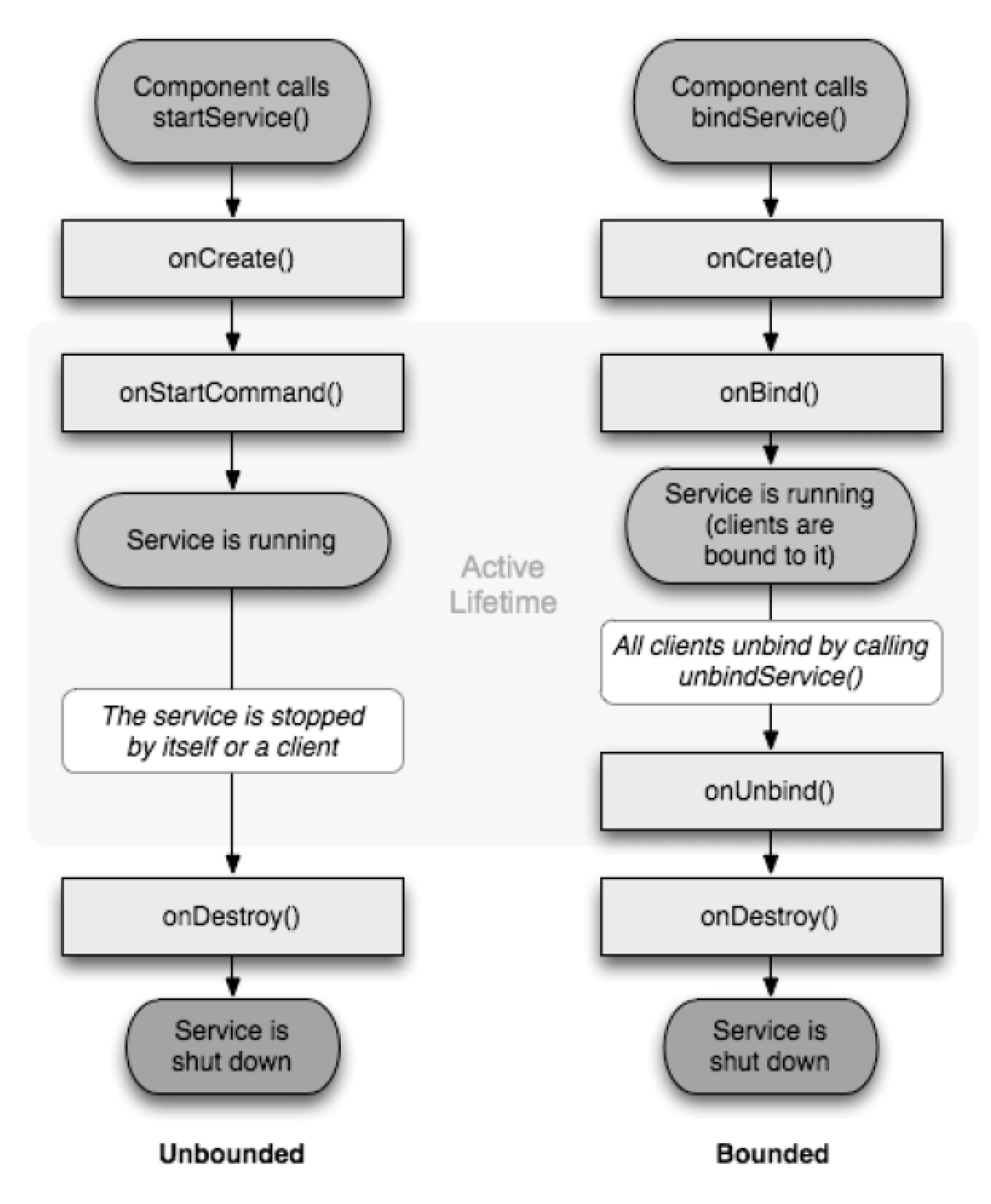


图 8-1　官方提供的 Service 的生命周期图

通过图 8-1 可以看出 Service 的生命周期并不向 Activity 或者 Fragment 那样复杂，只有 onCreate()、onStartCommand()、onDestroy()、onBind()、onUnbind()等几种。同时也可以看出 Service 的生命周期分为两种，正好应对着 Service 的两种启动方式，分别如表 8-4 所示。

表8-4　以start方式和bind方式开启服务时对应的生命周期

开启方式	生命周期
start 方式	onCreate()->onStartCommand()->Service running——调用 context.stopService() ->onDestroy()
bind 方式	onCreate()->onBind()->Service running——调用 context.unbindService()->onUnbind()->onDestroy()

具体来说，如果一个 Service 被某个 Activity 调用 Context.startService()方法启动，那么不管是否有 Activity 使用 bindService()绑定或 unbindService()解除绑定到 Service，该 Service 都在后台运行。同时，不管一个 Service 被 startService()方法启动几次，onCreate()方法也只会调用一次，onStart()方法将会被调用多次（对应调用 startService()的次数），并且系统只会创建 Service 的一个实例，所以我们应该知道只需要一次 stopService()方法的调用。该 Service 将会一直在后台运行，而不管对应程序的 Activity 是否在运行，直到被调用 stopService()或自身的 stopSelf()方法。当然如果系统资源不足，Android 系统也可能结束服务。

如果一个 Service 被某个 Activity 调用 Context.bindService()方法绑定启动，那么不管 bindService()方法被调用了几次，onCreate()方法都只会调用一次，同时 onStart()方法始终不会被调用。当连接建立之后，Service 将会一直运行，除非调用 Context.unbindService*()断开连接或者之前调用 bindService()的 Context 不存在了（如 Activity 被 finish 的时候），系统将会自动停止 Service，对应 onDestroy()将被调用。

在上面的两种生命周期中，我们可以看到，不管使用哪种方式，都会在创建时调用 onCreate()方法、在销毁时调用 onDestroy()方法。因此在开发中，我们会在 onCreate()方法中进行一些初始化的工作，而在 onDestroy()方法中进行一些清理工作。

另外，在前面提及同时以 start 方式和 bind 方式开启 Service 的状况。此时需要注意的是，需要同时调用 unbindService()与 stopService()才能终止 Service，而不管 unbindService()与 stopService()的调用顺序如何。先调用 unbindService()，此时服务不会自动终止，再调用 stopService()之后服务才会停止；先调用 stopService()，此时服务也不会终止，再调用 unbindService()或者之前调用 bindService()的 Context 不存在了（如 Activity 被 finish()的时候）之后服务才会自动停止。

8.2　Service 的简单实例

通过 8.1 节的学习，读者应该对 Service 有了一个全面的了解，也知道了创建与启动 Service 的具体步骤与方法。本节将通过几个实例带领读者一起学习如何使用 Service。

8.2.1 以 start 方式创建与启动 Service

通过 8.1 节的学习，我们知道了用 start 方式创建以及使用 Service 的 4 个步骤，下面我们按照这 4 个步骤来进行讲解。

步骤 01 创建一个继承 Service 的类 FirstService，在类中实现它的几个主要方法，为了验证 8.1 节所说的生命周期的结论，我们在生命周期方法中都加入了 Log。同时，为了模拟真实的开发环境，我们建立了一个线程，并在 onStartCommand(Intent intent, int flags, int startId)方法中使用这个线程，在 onDestroy()挂起线程，并销毁线程对象。具体的代码如下：

```
package com.buaa.service.service;

import android.app.Service;
import android.content.Intent;
import android.os.IBinder;
import android.util.Log;

public class FirstService extends Service {
    private Thread thread;
    private ServiceThread serviceThread;

    @Override
    public void onCreate() {
        super.onCreate();
        Log.i("service", "onCreate");
    }

    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        Log.i("service", "onStartCommand");
        serviceThread = new ServiceThread();
        thread = new Thread(serviceThread);
        //开启一个线程
        thread.start();
        return super.onStartCommand(intent, flags, startId);
    }

    @Override
    public void onDestroy() {
        super.onDestroy();
        //结束 run 方法的循环
        serviceThread.flag = false;
        //挂起线程
```

```
            thread.interrupt();
            thread = null;
            Log.i("service", "onDestroy");
        }

        @Override
        public IBinder onBind(Intent intent) {
            Log.i("service", "onBind");
            return null;
        }

        @Override
        public boolean onUnbind(Intent intent) {
            Log.i("service", "onUnbind");
            return super.onUnbind(intent);
        }

        class ServiceThread implements Runnable {
            //用 volatile 修饰保证变量在线程间的可见性
            volatile boolean flag = true;

            @Override
            public void run() {
                while (flag) {
                    try {
                        //间隔一秒
                        Thread.sleep(1000);
                    } catch (InterruptedException exception) {
                        Log.i("service", exception.toString());
                    }
                    Log.i("service", "thread  正在运行");
                }
            }
        }
    }
```

步骤 02 在 AndroidManifest.xml 中配置 Service。在讲解 Activity 时，我们讲解了显式意图和隐式意图的概念，不同形式的意图在 AndroidManifest.xml 中需要进行不同的配置，而且我们还解释了为何多使用隐式意图的方式进行 Activity 间跳转。这里特别说明一下，虽然在 Service 中也存在这样两种形式的意图，但是由于在 Android 5.0 之后 Google 公司出于安全考虑，禁止了隐式声明 Intent 来启动 Service。因此，在开发中建议只使用显式意图，配置如下（为防止一些新入门的读者放错位置，特别展示出全部配置文件）：

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.buaa.service">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".activity.MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <!-- 加入这样一条配置即可-->
        <service android:name=".service.FirstService"></service>
    </application>

</manifest>
```

步骤 03 通过 Context 调用 startService(Intent intent)方法来开启 Service、调用 stopService(Intent intent)方法来终止 Service。为了实现这样的目的，我们在 Activity 中使用两个按钮来分别负责 Service 的开启和终止。当然这只是一个实例，在开发实践中如何使用 Service 是十分灵活的，读者在开发时可以根据具体的需要来决定如何开启与终止 Service。实例代码如下所示。

Activity 的布局文件 activity_main.xml 代码：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".activity.MainActivity">

    <Button
        android:id="@+id/start_service"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="start service" />
```

```
    <Button
        android:id="@+id/stop_service"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="stop service" />
</LinearLayout>
```

Activity 代码如下：

```
package com.buaa.service.activity;

import android.content.Intent;
import android.support.v4.app.FragmentActivity;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;

import com.buaa.service.R;
import com.buaa.service.service.FirstService;

public class MainActivity extends AppCompatActivity implements View.OnClickListener {

    private Button startServiceButton;
    private Button stopServiceButton;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        initView();
    }

    private void initView() {
        startServiceButton = (Button) findViewById(R.id.start_service);
        stopServiceButton = (Button) findViewById(R.id.stop_service);
        startServiceButton.setOnClickListener(this);
        stopServiceButton.setOnClickListener(this);
    }

    @Override
    public void onClick(View v) {

        switch (v.getId()) {
```

```
                case R.id.start_service:
                    //创建意图
                    Intent startIntent = new Intent(this, FirstService.class);
                    //开启 Service
                    startService(startIntent);
                    break;
                case R.id.stop_service:
                    //创建意图
                    Intent stopIntent = new Intent(this, FirstService.class);
                    //终止服务
                    stopService(stopIntent);
                    break;
            }
        }
    }
```

在 Activity 中对 Button 按钮的点击事件进行了监听，并在点击时分别开启服务和终止服务。运行程序，效果如图 8-2 所示。

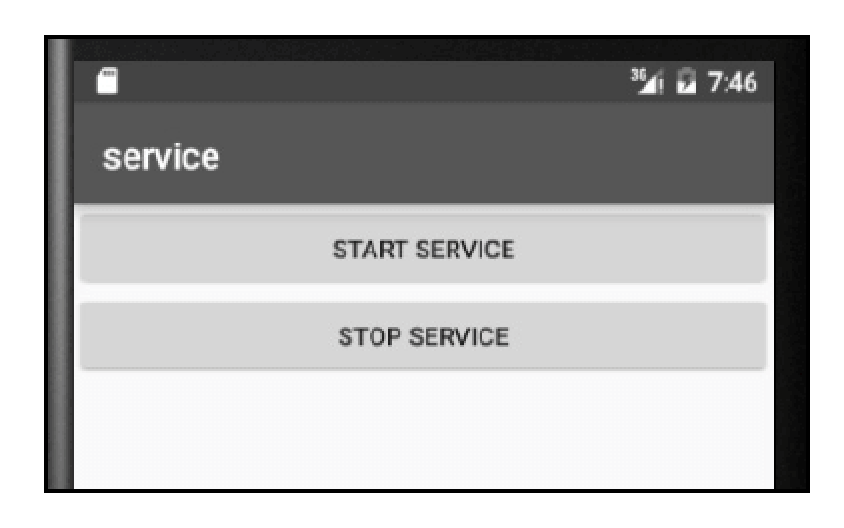


图 8-2　开启服务与停止服务

此时，点击“START SERVICE”按钮，观察 Log 的输出，发现 Service 以及 Service 中的线程确实被开启了，Log 如下：

```
11441-11441/com.buaa.service I/service: onCreate
11441-11441/com.buaa.service I/service: onStartCommand
11441-15211/com.buaa.service I/service: thread 正在运行
11441-15211/com.buaa.service I/service: thread 正在运行
11441-15211/com.buaa.service I/service: thread 正在运行
```

当点击“STOP SERVICE”按钮时，观察 Log 的输出，发现 Thread 不再运行，Service 也被终止了，Log 如下：

```
11441-15628/com.buaa.service I/service: thread 正在运行
11441-15628/com.buaa.service I/service: thread 正在运行
11441-15628/com.buaa.service I/service: thread 正在运行
11441-11441/com.buaa.service I/service: onDestroy
```

从输出的 Log 中还可以清晰看出以 start 方式开启的 Service 的生命周期恰是 8.1 节所阐述的那样，它的生命周期方法是按照 onCreate()→onStartCommand()→Service running——调用 context.stopService()→onDestroy()这样一条路径运行的。

此时如果开启了服务，但是并没有终止服务就直接退出程序，服务是不会被终止的，读者用上面的实例可以亲手进行测试。所以当开启服务之后，它是不会自己停止的，也不会随着开启者的销毁而销毁，所以一定要调用 stopService()方法来终止它，至于终止的时机需要根据具体的业务来把握。

8.2.2 以 bind 方式创建与绑定 Service

通过上一部分的学习，可以发现使用 start 方式开启 Service 非常简单，但是它有一个非常大的弊端，没有办法与开启者进行通信。为了解决这个问题，我们一般会使用 bind 方式来绑定 Service。以 bind 方式创建与绑定 Service 的步骤在 8.1 节也已讲述，此处将按照上述步骤用一个实例来进行讲解。

步骤 01 创建一个继承 Service 类的子类 SecondService 类，由于是采用 bind 方式绑定 Service，因此 SecondService 类与 FirstService 类有很大不同。

前文说过，采用此种方式绑定 Service 时会回调 onBind()方法，此方法会返回一个 IBind 类的对象。一般情况下，如果我们在某个 Activity 中绑定了此 Service，就会从 ServiceConnection 类 onServiceConnected(ComponentName name, IBinder service)方法中获取到 onBind()方法返回的 IBind 类的对象。Activity 与 Service 可以进行通信，利用的就是这个 IBind 类的对象。

所以，我们可以建立一个 IBind 类的子类，并在该类中封装 Service 对象，并在 onBind()方法中返回此类的对象。这样一来，在 Activity 中就可以对 Service 进行操作了。SecondService 类代码如下：

```
package com.buaa.service.service;

import android.app.Service;
import android.content.Intent;
import android.os.Binder;
import android.os.IBinder;
import android.util.Log;

public class SecondService extends Service {
    private String message;
    private boolean isRunning = true;
    private IBinder binder = new MyBinder();
    private SecondService.ServiceThread serviceThread;
    private Thread thread;

    @Override
    public IBinder onBind(Intent intent) {
        Log.i("service", "onBind");
        serviceThread = new ServiceThread();
        thread = new Thread(serviceThread);
```

```
        //开启一个线程
        thread.start();
        /**返回一个可以在 Activity 的 onServiceConnected()方法中接收到的 binder 对象
         *它是 Activity 和 Service 通信的桥梁
         *在 Activity 中通过这个 bind 对象可以得到 Service 的实例引用
         *通过获取的 Service 实例就可以调用相关方法和属性
         */
        return binder;
    }

    @Override
    public void onCreate() {
        Log.i("service", "onCreate");
        super.onCreate();
    }

    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        Log.i("service", "onStartCommand");
        return super.onStartCommand(intent, flags, startId);
    }

    @Override
    public boolean onUnbind(Intent intent) {
        Log.i("service", "onUnbind");
        return super.onUnbind(intent);
    }

    @Override
    public void onDestroy() {
        super.onDestroy();
        //结束 run 方法的循环
        serviceThread.flag = false;
        Log.i("service", "onDestroy");
    }

    class ServiceThread implements Runnable {
        //用 volatile 修饰保证变量在线程间的可见性
        volatile boolean flag = true;

        @Override
        public void run() {
            Log.i("service", "thread 开始运行");
```

```
                int i = 1;
                while (flag) {
                    if (mOnDataCallback != null) {
                        //通过线程模拟真实场景，循环改变数据
                        mOnDataCallback.onDataChange(message + i);
                    }
                    i++;
                    try {
                        Thread.sleep(1000);
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                }
            }
        }

        public class MyBinder extends Binder {
            public void setData(String message) {
                //从 Activity 传入 message 值
                SecondService.this.message = message;
            }

            public SecondService getService() {
                /**
                 *返回当前 SecondService 对象
                 *当 Activity 中获取 binder 类的实例后
                 *可以通过此方法获取 Service 类实例
                 */
                return SecondService.this;
            }
        }

        private OnDataCallback mOnDataCallback = null;

        public void setOnDataCallback(OnDataCallback mOnDataCallback) {
            this.mOnDataCallback = mOnDataCallback;
        }

        public interface OnDataCallback {
            void onDataChange(String message);
        }
    }
```

除了上面所说的 IBind 类之外，在此 Service 类中还建立了一个接口 OnDataCallback，用来进行数据的回调。

步骤 02 在 AndroidManifest.xml 中配置 Service 与前例相同，在 AndroidManifest.xml 中加入如下代码即可：

```
<service android:name=".service.SecondService"></service>
```

步骤 03 通过 Context 调用 bindService(Intent intent, ServiceConnection serviceConnection, int flags)方法来绑定 Service、调用 unBindService(ServiceConnection serviceConnection)方法来解绑 Service。

可以发现不管是绑定服务还是解绑服务都需传入一个 ServiceConnection 类的对象作为参数。ServiceConnection 类有两个比较重要的回调方法：onServiceConnected(ComponentName name, IBinder service)与 onServiceDisconnected(ComponentName name)，这两个方法中前者是当 Activity 与 Service 绑定时的回调方法，后者是解绑时的回调方法。一般情况下我们会在前一个方法中获取 IBinder 类的对象，并通过该对象获取 Service 类的实例，然后对 Service 进行操作，而在后一个方法中主要就是做一些清理性工作，比如销毁对象。

本实例中为了展现出 Service 与 Activity 真正在做通信，加入了一个 TextView，当 Activity 获取 Service 中的数据时，对该 TextView 的值进行修改。实例中布局文件在前例的 activity_main.xml 文件基础上修改，大同小异，代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".activity.MainActivity">

    <Button
        android:id="@+id/bind_service"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="bind service" />

    <Button
        android:id="@+id/unbind_service"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="unbind service" />

    <TextView
        android:id="@+id/text"
        android:layout_width="match_parent"
```

```
            android:layout_height="wrap_content"
            android:gravity="center"
            android:text="0"
            android:textSize="26sp" />
</LinearLayout>
```

MainActivity 的代码是本例的重点，为了方便读者理解加入了一些注释，代码如下：

```
package com.buaa.service.activity;

import android.content.ComponentName;
import android.content.Context;
import android.content.Intent;
import android.content.ServiceConnection;
import android.os.IBinder;
import android.support.v4.app.FragmentActivity;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;

import com.buaa.service.R;
import com.buaa.service.service.SecondService;

public class MainActivity extends AppCompatActivity implements View.OnClickListener {
    private Button bindServiceButton;
    private Button unbindServiceButton;
    private TextView textView;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        initView();
    }

    private void initView() {
        bindServiceButton = (Button) findViewById(R.id.bind_service);
        unbindServiceButton = (Button) findViewById(R.id.unbind_service);
        textView = (TextView) findViewById(R.id.text);
        bindServiceButton.setOnClickListener(this);
        unbindServiceButton.setOnClickListener(this);
    }
```

```
    @Override
    public void onClick(View v) {
        switch (v.getId()) {
            case R.id.bind_service:
                //创建意图
                Intent bindIntent = new Intent(this, SecondService.class);
                //绑定服务
                bindService(bindIntent, serviceConnection, Context.BIND_AUTO_CREATE);
                break;
            case R.id.unbind_service:
                //解绑服务
                unbindService(serviceConnection);
                break;
        }
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
    }

    private SecondService.MyBinder binder;
    private SecondService secondService;
    public ServiceConnection serviceConnection = new ServiceConnection() {
        @Override
        public void onServiceConnected(ComponentName name, IBinder service) {
            //得到 binder 实例
            binder = (SecondService.MyBinder) service;
            //给 Service 中的 message 设置一个值
            binder.setData("MainActivity：");
            //得到 Service 实例
            secondService = binder.getService();
            //设置接口回调获取 Service 中的数据
            secondService.setOnDataCallback(new SecondService.OnDataCallback() {
                @Override
                public void onDataChange(final String message) {
                    // 在非 UI 线程想要修改 UI 界面的内容时，使用此方法
                    runOnUiThread(new Runnable() {
                        @Override
                        public void run() {
                            textView.setText(message);
                        }
```

```
                        });
                    }
                });
            }

            @Override
            public void onServiceDisconnected(ComponentName name) {
                secondService = null;
            }
        };
}
```

结合注释，代码还是不难理解的。这里需要注意的是，修改 TextView 的值使用的 runOnUiThread(Thread thread)方法。有的读者可能会问，为何要在这个方法内修改 TextView 的值，而不是在 onDataChange(final String message)方法内直接修改。这涉及 Android 开发中的一个原则，那就是普通线程不可以修改 UI 界面，只有 UI 线程也就是 Activity 所在线程可以修改 UI 界面。如果在普通线程中需要修改 UI 界面了，那么只能使用 Android 中的消息处理机制，也就是我们常说的 Handler 机制。此处使用的 runOnUiThread(Runnable action)方法本质上就是 Handler 机制的应用。至于什么是 Handler 机制，下一节将详细讲解。

步骤 01 理解了代码之后，运行程序，效果如图 8-3 所示。

步骤 02 当点击“BIND SERVICE”按钮绑定完此服务后，会发现当前界面中的 TextView 被修改了，效果如图 8-4 所示。

步骤 03 当点击“UNBIND SERVICE”按钮解绑服务后，发现界面中 TextView 停止了变化，固定在了一个数字上，效果如图 8-5 所示。

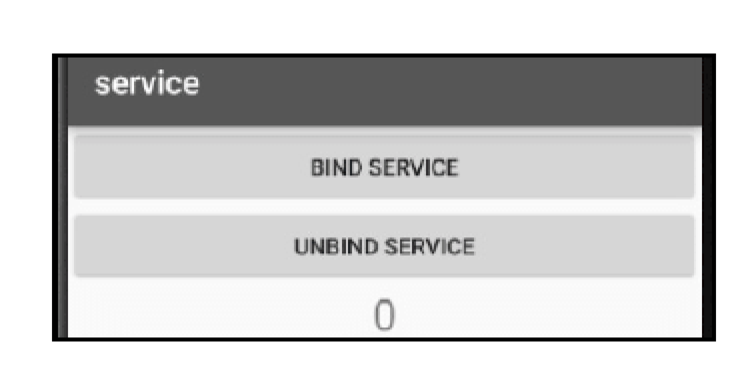


图 8-3 bind 服务与 unbind 服务

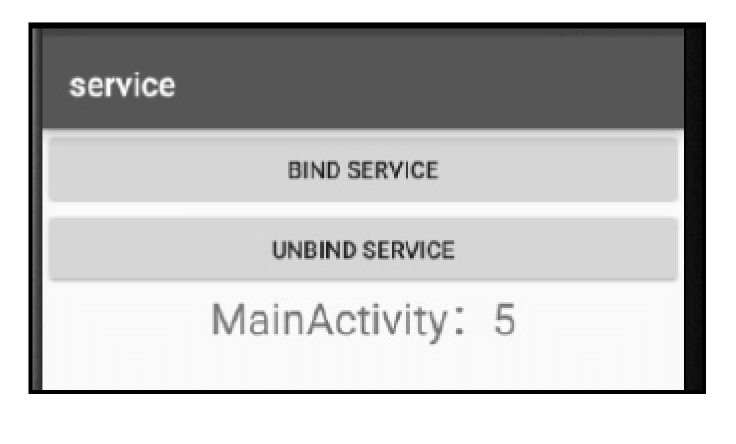


图 8-4 以 bind 方式运行的服务

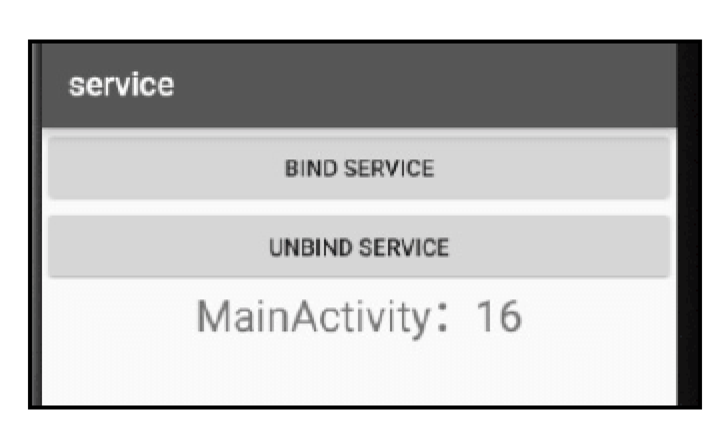


图 8-5 对以 bind 方式运行的服务进行 unbind 操作

此时观察 Log 的输出，效果如下：

```
7381-7381/com.buaa.service I/service: onCreate
7381-7381/com.buaa.service I/service: onBind
7381-10809/com.buaa.service I/service: thread 开始运行
7381-7381/com.buaa.service I/service: onUnbind
7381-7381/com.buaa.service I/service: onDestroy
```

发现 bind 方式绑定 Service 的生命周期正如 8.1 节所描述的那样，它的生命周期方法是按照

onCreate()→onBind()→Service running——调用 context.unbindService()→onUnbind()→onDestroy()这样的路径执行的。

8.3 Android 消息处理机制

Android 应用程序启动时，系统会创建一个主线程，负责与 UI 组件（widget、view）进行交互，比如控制 UI 界面显示、更新等；分发事件给 UI 界面处理，比如按键事件、触摸事件、屏幕绘图事件等，因此，Android 主线程也称为 UI 线程。

由此可知，UI 线程只能处理一些简单的、短暂的操作，如果要执行繁重的任务或者耗时很长的操作，比如访问网络、数据库、下载等，这种单线程模型会导致线程运行性能大大降低，甚至阻塞 UI 线程，如果被阻塞超过 5 秒，系统会提示应用程序无响应，也就是 ANR，这会直接导致退出整个应用程序或者短暂杀死应用程序。

除此之外，单线程模型的 UI 主线程也是不安全的，会造成不可确定的结果。线程不安全可以简单理解为：多线程访问资源时，有可能出现多个线程先后更改数据造成数据不一致。比如，A 工作线程（也称为子线程）访问某个公共 UI 资源，B 工作线程在某个时候也访问了该公共资源，当 B 线程正访问时，公共资源的属性已经被 A 改变了，这样 B 得到的结果不是所需要的，造成了数据不一致的混乱情况。

线程安全简单理解为：当一个线程访问功能资源时对该资源进行了保护，比如加了锁机制，当前线程在没有访问结束释放锁之前，其他线程只能等待直到释放锁才能访问，这样的线程就是安全的。

基于以上原因，Android 的单线程模型必须遵守两个规则：

（1）不要阻塞 UI 线程。

（2）不要在 UI 线程之外访问 UI 组件，即不能在子线程访问 UI 组件，只能在 UI 线程访问。

因此，Android 系统将大部分耗时、繁重任务交给子线程完成，不会在主线程中完成，解决了第一个难题；同时，Android 只允许主线程更新 UI 界面，子线程处理后的结果无法和主线程交互，即无法直接访问主线程，这就要用到 Handler 机制来解决此问题。

8.3.1 Handler 机制核心类介绍

在 Handler 机制中的所有故事都是围绕 Handler、Looper、Message 这 3 个类展开的。下面我们分别介绍一下这 3 个类。

1. Message

消息对象，顾名思义就是记录消息信息的类。Message 类有几个比较重要的字段，如表 8-5 所示。

表8-5　Message类的常用字段

字段名	作用
arg1	使用这个字段来传递整数类型的值，与 arg2 相同
arg2	使用这个字段来传递整数类型的值，与 arg1 相同
obj	这个字段是 Object 类型，可以通过这个字段传递某个复杂的消息内容到接收者中
what	这个字段是消息的标志，在消息处理中，可以根据这个字段区分消息，类似于在处理 button 事件时通过 switch(v.getId())判断是点击了哪个按钮

在使用 Message 时，可以通过 new Message()创建一个 Message 实例，但是 Android 官方更推荐我们通过 Message.obtain()或者 Handler.obtainMessage()获取 Message 对象。这并不一定是直接创建一个新的实例，而是先从消息池中看有没有可用的 Message 实例，存在则直接取出并返回这个实例。反之如果消息池中没有可用的 Message 实例，则根据给定的参数新建一个新 Message 对象。一般情况下，Android 系统默认情况下在消息池中实例化 10 个 Message 对象。

当 Message 实例被创建之后，使用 setDate()或者 arg 参数为 Message 携带一些数据，并通过 Handler 对象发送到 MessageQueue 中。

2. Looper

Looper 是 MessageQueue 的管理者。

MessageQueue 是一个消息队列，用来存放 Message 对象的数据结构，按照“先进先出”的原则存放消息。存放并非实际意义的保存，而是将 Message 对象以链表的方式串联起来的。MessageQueue 对象不需要自己创建，而是由 Looper 对象对其进行管理，一个线程最多只可以拥有一个 MessageQueue。可以通过 Looper.myQueue()获取当前线程中的 MessageQueue。

在一个线程中，如果存在 Looper 对象，则必定存在 MessageQueue 对象，并且只存在一个 Looper 对象和一个 MessageQueue 对象。在 Android 系统中，除了主线程有默认的 Looper 对象，其他线程默认是没有 Looper 对象的。如果想让新创建的线程拥有 Looper 对象，首先应调用 Looper.prepare()方法，然后调用 Looper.loop()方法。

另外，如果想要获取一个已经存在的 Looper 对象，可以通过 Looper.myLooper()获取，此外还可以通过 Looper.getMainLooper()获取当前应用系统中主线程的 Looper 对象。在这个地方有一点需要注意，假如 Looper 对象位于应用程序主线程中，那么 Looper.myLooper()和 Looper.getMainLooper()获取的是同一个对象。

3. Handler

消息的处理者。一般情况下，会在子线程中通过 Handler 对象把 Message 对象发送到 MessageQueue 中，然后在主线程中用该对象的 handleMessage(Message msg)方法接收 Message 对象，再对 UI 进行操作。

Handler 类的方法很多，常用的几种如表 8-6 所示。

表8-6 Handler类的常用方法

方法	作用
public final boolean sendEmptyMessage(int what)	用于发送一个只包含 what 值的 Message
public final boolean sendMessage(Message msg)	用于发送一个 Message
public final boolean sendMessageDelayed(Message msg, long nms)	用于延迟发送一个 Message，延迟时长为 nms（毫秒）
public final boolean hasMessages(int what)	用于判断消息队列中是否已经含有此 what 值的 Message
public final boolean post(Runnable r)	用于提交一个任务，并立即执行
public final boolean sendMessageDelayed(Runnable r，long nms)	用于提交一个延迟执行的任务，延迟时长为 nms（毫秒）

通过上面的学习，读者可能已经明白了 Handler 机制是如何发送消息到 MessageQueue 中的，但是对于 handleMessage(Message msg)方法为什么能够接收到 Message 还抱有疑问。其实，这里面的重点在于 Looper.loop()方法。此方法很重要，源码如下：

```
public static void loop() {
    final Looper me = myLooper();
    if (me == null) {
        throw new RuntimeException("No Looper; Looper.prepare() wasn't called on this thread.");
    }
    final MessageQueue queue = me.mQueue;

    // Make sure the identity of this thread is that of the local process,
    // and keep track of what that identity token actually is.
    Binder.clearCallingIdentity();
    final long ident = Binder.clearCallingIdentity();

    for (;;) {
        Message msg = queue.next(); // might block
        if (msg == null) {
            // No message indicates that the message queue is quitting.
            return;
        }

        // This must be in a local variable, in case a UI event sets the logger
        Printer logging = me.mLogging;
        if (logging != null) {
            logging.println(">>>>> Dispatching to " + msg.target + " " +
                    msg.callback + ": " + msg.what);
        }

        msg.target.dispatchMessage(msg);
```

```
            if (logging != null) {
                logging.println("<<<<< Finished to " + msg.target + " " + msg.callback);
            }

            // Make sure that during the course of dispatching the
            // identity of the thread wasn't corrupted.
            final long newIdent = Binder.clearCallingIdentity();
            if (ident != newIdent) {
                Log.wtf(TAG, "Thread identity changed from 0x"
                        + Long.toHexString(ident) + " to 0x"
                        + Long.toHexString(newIdent) + " while dispatching to "
                        + msg.target.getClass().getName() + " "
                        + msg.callback + " what=" + msg.what);
            }

            msg.recycleUnchecked();
        }
    }
```

通过源码分析，可以知道这是通过一个死循环不断地调用 MessageQueue 的 next()方法，这个 next()方法就是消息队列的出队方法。每当有一个消息出队，就将它传递到 msg.target 的 dispatchMessage(Message msg)方法中。dispatchMessage(Message msg)的源码如下：

```
    public void dispatchMessage(Message msg) {
        if (msg.callback != null) {
            handleCallback(msg);
        } else {
            if (mCallback != null) {
                if (mCallback.handleMessage(msg)) {
                    return;
                }
            }
            handleMessage(msg);
        }
    }
```

这样一来，handleMessage(Message msg)方法才可以获取到之前发送的消息。

另外，在 8.2 节中用到了 runOnUiThread(Thread thread)方法，当时说这其实使用的也是 Handler 机制。打开 runOnUiThread(Thread thread)方法，源码如下：

```
    public final void runOnUiThread(Runnable action) {
        if (Thread.currentThread() != mUiThread) {
            mHandler.post(action);
        } else {
```

```
            action.run();
        }
    }
```

通过分析发现，runOnUiThread(Thread thread)方法的逻辑很简单：如果当前的线程不等于UI 线程（主线程），就去调用 Handler 的 post()方法，否则直接调用 Runnable 对象的 run()方法。

8.3.2 Handler 机制使用实例

使用 Handler 机制，首先需要创建一个 Handler 对象，可以直接使用 Handler 无参构造函数创建 Handler 对象，或者是继承 Handler 类，重写 handleMessage(Message msg)方法来创建 Handler 对象。Google 官方提供了一个推荐的使用方式，代码如下：

```
class LooperThread extends Thread {
    public Handler mHandler;

    public void run() {
        Looper.prepare();

        mHandler = new Handler() {
            public void handleMessage(Message msg) {
                // process incoming messages here
            }
        };

        Looper.loop();
    }
}
```

通过上一部分的分析，读者应该能够很容易理解上面这种方式。但是在实际的开发实践中，大部分的 Handler 对象都是在主线程中创建的，此时已经存在了 Looper 对象，并不需要调用 Looper.prepare()与 Looper.loop()方法，直接构建一个 Handler 对象即可：

```
private Handler mHandler = new Handler() {
            public void handleMessage(Message msg) {
                // process incoming messages here
            }
        };
```

结合 8.2 节的例子，不使用 runOnUiThread(Thread thread)方法，而是使用 Handler 的sendMessage(Message msg)方法来发送消息，并使用 handleMessage(Message msg)方法来接收消息。在布局文件中添加一个 TextView，当程序运行 10 秒时修改 TextView 的内容。只需在布局文件中加入如下代码即可：

```
<TextView
        android:id="@+id/handler_text"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:gravity="center"
        android:text="handler"
        android:textSize="26sp" />
```

在 Activity 中，创建 Handler，并使用 handler 发送消息，代码如下：

```
package com.buaa.service.activity;

import android.content.ComponentName;
import android.content.Context;
import android.content.Intent;
import android.content.ServiceConnection;
import android.os.Handler;
import android.os.IBinder;
import android.os.Message;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;
import android.widget.Toast;

import com.buaa.service.R;
import com.buaa.service.service.SecondService;

public class MainActivity extends AppCompatActivity implements View.OnClickListener {
    private Button bindServiceButton;
    private Button unbindServiceButton;
    private TextView textView;
    private TextView handlerTextView;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        initView();
    }

    private void initView() {
        bindServiceButton = (Button) findViewById(R.id.bind_service);
```

```
        unbindServiceButton = (Button) findViewById(R.id.unbind_service);
        textView = (TextView) findViewById(R.id.text);
        handlerTextView = (TextView) findViewById(R.id.handler_text);
        bindServiceButton.setOnClickListener(this);
        unbindServiceButton.setOnClickListener(this);
    }

    @Override
    public void onClick(View v) {
        switch (v.getId()) {
            case R.id.bind_service:
                //创建意图
                Intent bindIntent = new Intent(this, SecondService.class);
                //绑定服务
                bindService(bindIntent, serviceConnection, Context.BIND_AUTO_CREATE);
                break;
            case R.id.unbind_service:
                //创建意图
                Intent unbindIntent = new Intent(this, SecondService.class);
                //解绑服务
                unbindService(serviceConnection);
                break;
        }
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
        unbindService(serviceConnection);
    }

    private SecondService.MyBinder binder;
    private SecondService secondService;
    public ServiceConnection serviceConnection = new ServiceConnection() {
        @Override
        public void onServiceConnected(ComponentName name, IBinder service) {
            //得到 binder 实例
            binder = (SecondService.MyBinder) service;
            //给 Service 中的 message 设置一个值
            binder.setData("MainActivity：");
            //得到 Service 实例
            secondService = binder.getService();
```

```
            //设置接口回调获取 Service 中的数据
            secondService.setOnDataCallback(new SecondService.OnDataCallback() {
                @Override
                public void onDataChange(final String message) {
                    if (message.equals("MainActivity：" + 10)) {
                        Bundle bundle = new Bundle();
                        bundle.putString("name", "李瑞奇");
                        Message msg1 = mHandler.obtainMessage();
                        msg1.setData(bundle);
                        msg1.what = 1;
                        msg1.arg1 = 26;
                        mHandler.sendMessage(msg1);
                    } else if (message.equals("MainActivity：" + 20)) {
                        mHandler.sendEmptyMessage(2);
                    } else {
                        mHandler.post(new Runnable() {
                            @Override
                            public void run() {
                                textView.setText(message);
                            }
                        });
                    }
                }
            });
        }

        @Override
        public void onServiceDisconnected(ComponentName name) {
            secondService = null;
        }
    };

    private Handler mHandler = new Handler() {
        public void handleMessage(Message msg) {
            switch (msg.what) {
                case 1:
                    String name = (String) msg.getData().get("name");
                    int age = msg.arg1;
                    handlerTextView.setText(name + "的年纪为：" + age);
                    break;
                case 2:
                    Toast.makeText(MainActivity.this, "发送的一个空的 Message",
```

```
                    Toast.LENGTH_LONG).show();
                    break;
            }
        }
    };
}
```

这里分别使用 mHandler.sendMessage(msg1)、 mHandler.sendEmptyMessage(2)两种方式来发送消息，并在 handleMessage(Message msg)中用 switch (msg.what) 来分别获取。同时还用了 mHandler.post(new Runnable(){})这样一个与 runOnUiThread(Thread thread)同样效果的方法。Handler 的其他几个方法与上述 3 种用法一致，稍加改动就可以使用，此处不再叙述。

运行程序，绑定服务之后，当第 10 秒之后可以看到 TextView 的值被修改，如图 8-6 所示。第 20 秒之后，Toast 出现，效果如图 8-7 所示。





图 8-6 使用 sendMessage 方法来发送消息



图 8-7 使用 post 方法来发送消息 I

8.3.3 Handler 机制与 AsyncTask 比较分析

AsyncTask 是 Android 提供的轻量级的异步类，可以直接继承 AsyncTask，在类中实现异步操作，并提供接口反馈当前异步执行的程度（可以通过接口实现 UI 进度更新），最后反馈执行的结果给 UI 主线程。这个类的设计目的很明确，就是为了“执行一个较为耗时的异步任务（最多几秒钟），然后更新界面”。

这种需求本可以使用 Handler 和 Thread 来实现，但是在单个后台异步处理时显得代码过多、结构过于复杂，因此 Android 提供了 AsyncTask 类。但是在使用多个后台异步操作并需要进行 UI 变更时，使用 AsyncTask 类就变得复杂起来，使用 Handler 和 Thread 则更加合适。

另外，这里所说的轻量级只是代码上的轻量级，而非性能上的，使用 AsyncTask 会更加消耗性能。

AsyncTask 类有 4 个重要方法，这也是当一个异步任务被执行时要经历的 4 步，如表 8-7 所示。

表8-7　AsyncTask类的4个重要方法

方法	作用
onPreExecute()	在异步任务开始执行前在 UI 线程中执行，一般用来设置任务参数
doInBackground()	最重要的方法，在子线程中执行（事实上，只有它在子线程中执行，其他方法都在 UI 线程中执行）。当 onPreExecute()结束后，本方法立刻执行，用来进行后台的耗时计算，异步任务的参数会被传给它，执行完成的结果会被送给第四步。执行途中，它还可以调用 publishProgress()方法来通知 UI 线程当前执行的进度
onProgressUpdate()	当 publishProgress()被调用后，它在 UI 线程中执行，刷新任务进度，一般用来刷新进度条等 UI 部件
onPostExecute()	当后台的异步任务完成后，它会在 UI 线程中被调用，并获取异步任务执行完成的结果

下面用一个实例来讲解如何使用 AsyncTask 类。创建一个继承自 AsyncTask 类的 MyAsyncTask 类，实现它的 4 个主要方法，并创建一个带参数的构造方法，用以介绍 Activity 类的 Content 和布局管理器。在 doInBackground()方法中模拟下载任务，并每隔 1 秒更新一次进度条。代码如下：

```
package com.buaa.service.util;

import android.app.ProgressDialog;
import android.content.Context;
import android.os.AsyncTask;
import android.view.ViewGroup;
import android.widget.TextView;

public class MyAsyncTask extends AsyncTask {
    private ProgressDialog progressDialog;
    private ViewGroup viewGroup;
    private Context context;

    public MyAsyncTask(Context context, ViewGroup viewGroup) {
        this.viewGroup = viewGroup;
        this.context = context;
    }

    @Override
    protected void onPreExecute() {
        super.onPreExecute();
        //使用一个进度条对话框
        progressDialog = new ProgressDialog(context);
        progressDialog.setTitle("正在下载中，请稍后......");
        //设置 ProgressDialog 样式为圆圈的形式
        progressDialog.setProgressStyle(ProgressDialog.STYLE_HORIZONTAL);
        progressDialog.show();
```

```
        }

        @Override
        protected String doInBackground(Object[] params) {
            //execute()方法传递的参数
            String address = (String) params[0];
            int flag = 0;
            while (flag < 100) {
                flag += 20;
                //更新进度，将回调 onProgressUpdate()方法
                publishProgress(flag);
                try {
                    Thread.sleep(1000);
                } catch (Exception e) {

                }
            }
            return address + ":从这个下载地址下载了一本小说，欢迎阅读。";
        }

        @Override
        protected void onProgressUpdate(Object[] values) {
            super.onProgressUpdate(values);
            //更新进度条
            progressDialog.setProgress((Integer) values[0]);
        }

        @Override
        protected void onPostExecute(Object o) {
            super.onPostExecute(o);
            //在布局中加入一个 TextView
            TextView textView = new TextView(context);
            textView.setText((String) o);
            viewGroup.addView(textView);
            //关闭进度条
            progressDialog.dismiss();
        }
    }
```

同时建立一个 Activity，在布局文件中加入一个 Button，当点击 Button 时，实例化 MyAsyncTask 类并调用 execute()方法。布局文件代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:id="@+id/line"
    tools:context="com.buaa.service.activity.AsyncTaskActivity">

    <Button
        android:id="@+id/download"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="点击下载" />
</LinearLayout>
```

Activity 类的代码如下：

```
package com.buaa.service.activity;

import android.app.ProgressDialog;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.LinearLayout;

import com.buaa.service.R;
import com.buaa.service.util.MyAsyncTask;

public class AsyncTaskActivity extends AppCompatActivity implements View.OnClickListener {
    private Button download;
    private ProgressDialog progressDialog;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_async_task);
        initView();
    }

    private void initView() {
        download = (Button) findViewById(R.id.download);
        download.setOnClickListener(this);
```

```
        }

        @Override
        public void onClick(View v) {
            MyAsyncTask myAsyncTask = new MyAsyncTask(this, (LinearLayout) findViewById(R.id.line));
            myAsyncTask.execute("小说下载地址");
        }
    }
```

运行程序并点击“下载”按钮，将出现一个提示下载的进度条，如图 8-8 所示。

当完成下载之后，UI 界面中会显示刚才模拟下载的文本，效果如图 8-9 所示。



图 8-8　点击下载并使用 progressDialog 来显示下载进度



图 8-9　将下载的内容传递到 UI 线程并显示

8.4　前台服务

一个 Service 不管是被启动或是被绑定，默认是运行在后台的。有一种特殊的服务叫前台服务，是一种能被用户意识到它存在的服务，默认是不会被系统自动销毁的，但是必须提供一个状态栏 Notification，在通知栏放置一个持续的标题。这个 Notification 是不能被忽略的，除非服务被停止或从前台删除。这类服务主要用于一些需要用户能意识到它在后台运行并且随时可以操作的业务，如音乐播放器，设置为前台服务，使用一个 Notification 显示在通知栏，可以使用户切歌或是暂停之类的。

前台服务与普通服务的定义规则是一样的，也需要继承 Service，这里没有区别，唯一的区别是在服务里需要使用 Service.startFroeground(int id,Notification notification)方法设置当前服务为一个前台服务，并为其制定 Notification。其中的参数 id 是一个唯一标识通知的整数，但是这里注意这个整数一定不能为 0，notification 为前台服务的通知，并且这个 notification 对象只需要使用 startForeground()方法设置即可。前台服务可以通过调用 stopForeground(true)来使当前服务退出前台，但是并不会停止服务。

有一点需要注意，startForeground()需要在 Android 2.0 之后的版本才生效，在这之前的版本使用 setForeground()来设置前台服务，并且需要 NotificationManager 对象来管理通知，但是现在市面上基本上已经很少有 2.0 或以下的设备了，所以也不用太在意。

通过上面的介绍，会发现前台服务和 Notification 具有很强的关联，所以在讲解前台服务之前先对 Notification 进行简单的介绍，关于通知的更多内容，将在多媒体一章进行讲解。

8.4.1 Notification 简介与使用

因为 Android 的快速发展（版本快速升级）出现了一些兼容性的问题。对于 Notification 而言，Android 3.0 是一个分水岭，在其之前构建 Notification 推荐使用 NotificationCompate.Builder，是一个 Android 向下版本的兼容包，而在 Android 3.0 之后，一般推荐使用 NotificationCompat.Builder 方式构建。而现在主流的版本是 5.0 和 6.0，所以本文将使用 NotificationCompat.Builder 方式构建 Notification。对于一个简单的通知，只需要设置下面几个属性即可：

- 小图标，使用 setSamllIcon()方法设置。
- 标题，使用 setContentTitle()方法设置。
- 文本内容，使用 setContentText()方法设置。

当然，在使用通知时，一般情况下点击该通知能够执行指定的意图，这是使用 PendingIntent 类来实现的，详细的内容也将在多媒体一章的 Notification 一节中讲述，这里只要能够看懂即可。

下面通过一个简单的实例让读者能够更直观地感受 Notification 是如何使用的。新建一个 Activity，代码如下：

```
package com.buaa.service.activity;

import android.app.Notification;
import android.app.NotificationManager;
import android.app.PendingIntent;
import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;

import com.buaa.service.R;

public class ForeActivity extends AppCompatActivity {

    private Notification notification;
    private NotificationManager notificationManager;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
```

```
            super.onCreate(savedInstanceState);
            setContentView(R.layout.activity_fore);

            initView();
        }

        private void initView() {
            NotificationCompat.Builder builder = new NotificationCompat.Builder(this);
            // 实例化一个意图，当点击通知时，会跳转执行这个意图
            Intent intent = new Intent(this, ForeActivity.class);
            //将 intent 意图进行封装
            PendingIntent pendingIntent = PendingIntent.getActivity(this, 0, intent, PendingIntent.
             FLAG_CANCEL_CURREN);
            //设置 Notification 的点击之后执行的意图
            builder.setContentIntent(pendingIntent);
            builder.setSmallIcon(R.drawable.ic_launcher);
            builder.setContentTitle("酷我音乐");
            builder.setContentText("正在播放的歌曲：仰望星空");
            notification = builder.build();

            //实例化一个 NotificationManager
            notificationManager = (NotificationManager) getSystemService(NOTIFICATION_SERVICE);
            //使用 NotificationManager 来打开 notification
            notificationManager.notify(0, notification);
        }
    }
```

运行程序，打开通知栏，会出现一条通知，效果如图 8-10 所示。点击此通知将会跳转到 ForeActivity 中。



图 8-10　Notification 的简单使用

8.4.2　前台服务使用实例

在了解了 Notification 之后，前台服务的使用就变得简单了，只需新建一个 Notification 并使用 startForeground(int id, Notification notification)方法打卡即可，当然在服务销毁时也需要使用 stopForeground(true)来停止前台服务。下面通过一个实例来学习如何使用前台服务。实例中包含一个 Service 类、一个 Activity 类以及一个 Activity 类对应的布局文件。此 Activity 类和布局文件与 8.2 节的第二个实例大致相当，仅仅是通过两个 Button 按钮来操作绑定服务和解绑服务的操作，布局文件中也只包括两个 Button 按钮和一个文本框。PlayActivity 类代码如下：

```
package com.buaa.service.activity;

import android.content.Intent;
```

```
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;

import com.buaa.service.R;
import com.buaa.service.service.ForegroundService;

public class PlayActivity extends AppCompatActivity implements View.OnClickListener {
    private Button playSongButton;
    private Button stopSongButton;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_fore);
        initView();
    }

    private void initView() {
        playSongButton = (Button) findViewById(R.id.play_song);
        stopSongButton = (Button) findViewById(R.id.stop_song);
        playSongButton.setOnClickListener(this);
        stopSongButton.setOnClickListener(this);
    }

    @Override
    public void onClick(View v) {
        switch (v.getId()) {
            case R.id.play_song:
                Intent startIntent = new Intent(this, ForegroundService.class);
                startService(startIntent);
                break;
            case R.id.stop_song:
                Intent stopIntent = new Intent(this, ForegroundService.class);
                stopService(stopIntent);
                break;
        }
    }
}
```

布局文件 activity_fore.xml 的代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <TextView
        android:id="@+id/song_name"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="正在播放的歌曲：仰望星空"
        android:textSize="24sp" />

    <Button
        android:id="@+id/play_song"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="点击播放音乐" />

    <Button
        android:id="@+id/stop_song"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="结束播放音乐" />

</LinearLayout>
```

Service 类的子类 ForegroundService 类也和之前的代码相似，只是多了一个 Notification 而已，代码如下：

```
package com.buaa.service.service;

import android.app.Notification;
import android.app.PendingIntent;
import android.app.Service;
import android.content.Intent;
import android.os.IBinder;

import com.buaa.service.R;
import com.buaa.service.activity.PlayActivity;

public class ForegroundService extends Service {
    private Notification notification;
```

```
    @Override
    public void onCreate() {
        super.onCreate();
        buildDialog();
    }

    @Override
    public void onDestroy() {
        super.onDestroy();
        stopForeground(true);
    }

    @Override
    public IBinder onBind(Intent intent) {
        return null;
    }

    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        buildDialog();
        play();
        startForeground(1, notification);
        return super.onStartCommand(intent, flags, startId);
    }

    private void play() {
        //使用多线程播放音乐
    }

    private void buildDialog() {
        NotificationCompat.Builder builder = new NotificationCompat.Builder(this);
        // 实例化一个意图，当点击通知时会跳转执行这个意图
        Intent intent = new Intent(this, PlayActivity.class);
        //将 intent 意图进行封装
        PendingIntent pendingIntent = PendingIntent.getActivity(this, 0, intent,
        PendingIntent.FLAG_CANCEL_CURREN);
        //设置 Notification 的点击之后执行的意图
        builder.setContentIntent(pendingIntent);
        builder.setSmallIcon(R.drawable.ic_launcher);
        builder.setContentTitle("酷我音乐");
        builder.setContentText("正在播放的歌曲：仰望星空");
         //必须设置
        builder.setOngoing(true);
```

```
                notification = builder.build();
            }
        }
```

实例的代码逻辑相当简单，和之前的实例并无本质区别。运行程序，点击“播放音乐”就会在通知栏出现一个通知，点击“结束音乐播放”按钮时结束服务，通知也随之消失。效果如图 8-11 所示。



图 8-11　使用 Notification 实现前台服务

看到这样的效果，可能部分读者会疑惑，因为从单纯的界面看这与之前单独使用 Notification 的效果并无区别。这里必须指出，虽然显示的相同，但是使用前台服务实现的效果可以使服务因存在时间较长而被 Android 系统杀死。

8.5　IntentService

通过前面几节的介绍，读者会发现我们在使用 Service 时总会创建一个线程来执行任务，而不是直接在 Service 中执行。这是因为 Service 中的程序仍然运行于主线程中，当执行一项耗时操作时，不新建一个线程的话很容易导致 Application Not Responding 错误。当需要与 UI 线程进行交互时，使用 Handler 机制来进行处理。

为了简化操作，Android 提供了 IntentService 类。IntentService 是 Android 中提供的后台服务类，是 Service 自动实现多线程的子类。IntentService 在 onCreate()函数中通过 HandlerThread 单独开启一个线程来处理所有 Intent 请求对象所对应的任务，这样以免请求处理阻塞主线程。执行完一个 Intent 请求对象所对应的工作之后，如果没有新的 Intent 请求到达，就自动停止 Service；否则执行下一个 Intent 请求所对应的任务，直至最后执行完队列的所有命令，服务也随即停止并被销毁。所以如果使用 IntentService，用户并不需要主动使用 stopService() 或者在 IntentService 中使用 stopSelf()来停止。

IntentService 在处理请求时采用的也是 Handler 机制，它通过创建一个名叫 ServiceHandler 的内部 Handler 直接绑定到 HandlerThread 所对应的子线程。ServiceHandler 把处理一个 intent 所对应的请求都封装到 onHandleIntent()方法中，在开发时只需要直接重写 onHandleIntent()方法，当开启服务之后系统会自动调用此方法来处理请求。

使用 IntentService 相当简单，只需继承 IntentService 类，实现 onHandleIntent()方法并在其中处理相关请求的操作即可。下面通过一个实例来说明。

创建一个 Activity 类，并在布局文件中加入一个 Button 来开启服务。代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
```

```
    tools:context="com.buaa.service.activity.IntentServiceActivity">

    <Button
        android:id="@+id/down"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="下载文件" />
</RelativeLayout>
```

在 Activity 类中捕获 Button 按钮的点击事件，开启服务，代码如下：

```
package com.buaa.service.activity;

import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;

import com.buaa.service.R;
import com.buaa.service.service.MyIntentService;

public class IntentServiceActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_intent_service);

        Button button = (Button) findViewById(R.id.down);
        button.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent intent = new Intent(IntentServiceActivity.this, MyIntentService.class);
                intent.setAction("ServiceAction");
                intent.putExtra("path", "www.baiud.comm");
                IntentServiceActivity.this.startService(intent);
            }
        });
    }
}
```

最后创建一个类继承 IntentService，代码如下：

```
package com.buaa.service.service;

import android.app.IntentService;
import android.content.Intent;
import android.util.Log;

public class MyIntentService extends IntentService {
    public MyIntentService() {
        super("MyIntentService");
    }

    @Override
    protected void onHandleIntent(Intent intent) {
        if (intent != null) {
            final String action = intent.getAction();
            if ("ServiceAction".equals(action)) {
                final String path = intent.getStringExtra("path");
                handleDownload(path);
            }
        }
    }

    private void handleDownload(String path) {
        try {
            //模拟上传耗时
            Thread.sleep(3000);
            Log.i("MyIntentService", "从地址为： " + path + "的网站下载了一部小说");
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }

    @Override
    public void onCreate() {
        super.onCreate();
        Log.i("MyIntentService", "服务被开启");
    }

    @Override
    public void onDestroy() {
        super.onDestroy();
        Log.i("MyIntentService", "服务被杀死");
    }
}
```

在 AndroidManifest.xml 文件中注册此服务：

```
<service android:name=".service.MyIntentService" />
```

运行程序，点击按钮，观察 Log 会发现，确实如前文所说，当 Intent 请求的操作完成之后，Service 会自动销毁。Log 如下:

```
2393-2393/com.buaa.service I/MyIntentService: 服务被开启
2393-2560/com.buaa.service I/MyIntentService: 从地址为：www.baiud.comm的网站下载了一部小说
2393-2393/com.buaa.service I/MyIntentService: 服务被杀死
```

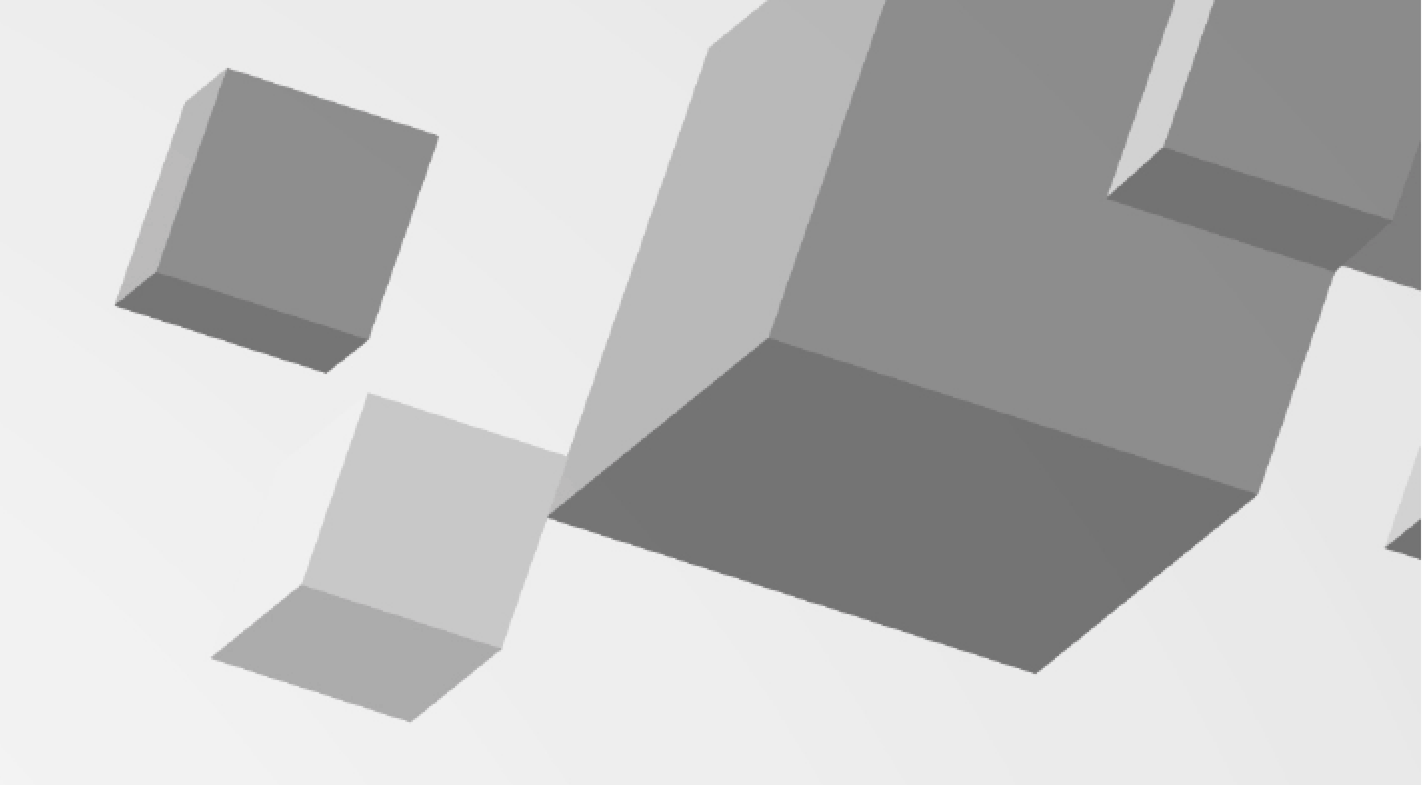
多次点击按钮，通过 Log 可以发现，此时 Service 会依次执行这些请求，直至所有请求处理完成之后 Service 自动销毁。Log 如下：

```
2393-2393/com.buaa.service I/MyIntentService: 服务被开启
2393-5323/com.buaa.service I/MyIntentService: 从地址为：www.baiud.comm的网站下载了一部小说
2393-5323/com.buaa.service I/MyIntentService: 从地址为：www.baiud.comm的网站下载了一部小说
2393-5323/com.buaa.service I/MyIntentService: 从地址为：www.baiud.comm的网站下载了一部小说
2393-2393/com.buaa.service I/MyIntentService: 服务被杀死
```

实例很简单、很容易理解，通过这个实例可以很容易地学会如何使用 IntentService。当然如果读者想要进一步了解 IntentService 的运行机制，也可以阅读 IntentService 类的源码。

8.6 小　　结

本章系统地讲述 Service 是什么、Service 的分类、为什么需要使用 Service 以及 Service 的几种使用方法，并结合 Service 讲解了 Handler 机制，同时还简单介绍了 AsyncTask 的用法。本章的内容相对比较重要，尤其是 Handler 机制的相关内容，读者需要认真理解，多做试验。



第 9 章 Android 广播机制

玩过收音机的人都听过广播，例如中央人民广播电台每天都会在固定时间广播固定的节目。在计算机领域，网络通信技术也使用广播，广播数据包会被发送到同一网络上的所有端口，这样在该网络中的每台主机都将会收到这条广播。在 Android 中，为了便于进行系统级别的消息通知，Android 也引入了一套类似的广播消息机制。本章将就 Android 中的广播机制进行详细讲解。

9.1 广播机制概述

广播（Broadcast）是一种广泛用于应用程序之间传递消息的机制，是 Android 系统的四大组件之一。广播机制包含 3 个基本要素：广播（Broadcast），用于发送广播；广播接收器（BroadcastReceiver），用于接收广播；意图内容（Intent），用于保存广播相关信息的媒介。

广播分为两个方面：广播发送者和广播接收者（Broadcast Receiver），在 Android 系统中很多操作完成以后都会发送广播，比如说发送短信息、打出一个电话、开机或者网络状态改变和电量改变等。如果某些应用程序想要在这些操作完成以后做一些相应的处理，就可以对这些广播做接收。这个广播跟传统意义中的电台广播有些相似，只是传统电台广播发送的是语音，而 Android 系统发送的是目的意图 Intent。之所以叫广播，就是因为它与传统的广播很相似，只负责播放而不管接收者“听不听”，也就不管接收方如何处理。

Android 中的每个应用程序都可以对自己需要的广播进行注册，这样该程序就可以接收到自己需要的广播内容，这些广播可能是来自于系统的，也可能是来自于其他应用程序的。Android 提供了一套完整的 API，允许应用程序自由地发送和接收广播。

Android 中的广播按照发送类型可以分为两种：普通广播和有序广播。

- 普通广播（Normal broadcasts）是一种完全异步执行的广播，效率较高，在广播发出之后，所有的广播接收者甚至可能会在同一时刻接收到这条广播消息，因此它们之间没有任何先后顺序可言。
- 有序广播（Ordered broadcasts）则是一种同步执行的广播，在广播发出之后，同一时刻只会有一个广播接收者能够收到这条广播消息，当这个广播接收者中的逻辑执行完毕后，广播才会继续传递。广播接收者是有先后顺序的，优先级高的广播接收者可以先收到广播消息，并且前面的广播接收者还可以截断正在传递的广播，使后面的广播接收者无法收到广播消息。

在开发中，广播一般会在下面几种情况下使用：

- 同一 App 内部的同一组件内的消息通信（单个或多个线程之间）。
- 同一 App 内部的不同组件之间的消息通信（单个进程）。
- 同一 App 具有多个进程的不同组件之间的消息通信。
- 不同 App 组件之间的消息通信。
- Android 系统在特定情况下与 App 之间的消息通信。

在这里我们可以看到 Broadcast 也可以在不同 App 应用之间进行消息通信。如果我们开发一个应用就需要在允许的情况下自动填充短信中的验证码，那么这时要监听用户短信，短信和自己的 App 就处在不同的进程之间。Activity 和 Service 在某些情况下的通信也可以借助 Broadcast，这时就是在同一进程不同组件之间的消息通信。

另外，需要注意的是，当我们通过广播接收者处理相应的广播时，不推荐进行任何耗时操作，因为在广播接收器中是不允许开启线程的，当 onReceive()方法运行了较长时间而没有结束时，程序就会报错。因此广播接收器更多的是扮演一种打开程序其他组件的角色，比如创

建一条状态栏通知或者启动一个服务等。

9.2 使用系统广播

Android 内置了很多系统级别的广播，我们可以在应用程序中通过监听这些广播来得到各种系统的状态信息，比如手机开机完成后会发出一条广播、电池的电量发生变化会发出一条广播、时间或时区发生改变也会发出一条广播等。

想要实现广播的接收，就需要使用广播接收者。广播接收者可以自由地对自己感兴趣的广播进行注册，当有相应的广播发出时，广播接收者就能够收到该广播，并在内部处理相应的逻辑。注册广播的方式一般有两种，在代码中注册和在 AndroidManifest.xml 中注册，其中前者被称为动态注册，后者被称为静态注册。下面分别通过检测电话状态（动态注册）以及应用开机启动（静态注册）两个实例来讲解系统广播的具体用法。

9.2.1 动态注册广播实例

这里通过一个检测电话状态的实例来讲解如何动态注册广播。在实例中如果想要接收到这些电话状态的广播就需要使用广播接收者。实现一个广播接收者只需要新建一个继承自 BroadcastReceiver 类的子类并重写父类的 onReceive()方法就行了。这样当有广播到来时，onReceive()方法就会得到执行，具体的业务逻辑只需在 onReceive()方法中处理就可以了。先新建一个继承自 BroadcastReceiver 类的 CallBroadCast 类，代码如下：

```
package com.buaa.broadcast.broadcast;

import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.widget.Toast;

public class CallReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        Toast.makeText(context, "您拨打了电话", Toast.LENGTH_LONG).show();
    }
}
```

可以看到，CallReceiver 类是继承自 BroadcastReceiver 的，并重写了父类的 onReceive()方法。这样每当电话状态发生变化时，onReceive()方法就会得到执行，这里只是简单地使用 Toast 提示了一段文本信息。

然后新建一个 Activity 类，动态注册广播。当然这里需要注意权限问题，读取电话状态是一个危险权限，除了要在 AndroidManifest.xml 文件中加入如下代码：

```
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
```

```
<uses-permission android:name="android.permission.PROCESS_OUTGOING_CALLS"/>
```

还需要动态获取相关权限：

```
package com.buaa.broadcast.activity;

import android.Manifest;
import android.content.IntentFilter;
import android.content.pm.PackageManager;
import android.os.Build;
import android.support.v4.app.ActivityCompat;
import android.support.v4.content.ContextCompat;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.Toast;

import com.buaa.broadcast.R;
import com.buaa.broadcast.broadcast.CallReceiver;

public class MainActivity extends AppCompatActivity {

    private CallReceiver callReceiver;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        getPermission();
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
        unregisterReceiver(callReceiver);
    }

    private void setCallReceiver() {
        callReceiver = new CallReceiver();
        IntentFilter filter = new IntentFilter();
        filter.addAction("android.intent.action.PHONE_STATE");
        filter.addAction("android.intent.action.NEW_OUTGOING_CALL");
        registerReceiver(callReceiver, filter);
    }
```

```
public void getPermission() {
    //判断版本号，在 api23 也就是 6.0 版本之前能直接获得权限
    if (Build.VERSION.SDK_INT >= 23) {
        int checkCALLPermission = ContextCompat.
                checkSelfPermission(this,
                        Manifest.permission.READ_PHONE_STATE);
        int checkOutCALLPermission = ContextCompat.
                checkSelfPermission(this,
                        Manifest.permission.PROCESS_OUTGOING_CALLS);
        //判断是否具有权限
        if (checkCALLPermission != PackageManager.PERMISSION_GRANTED ||
                checkOutCALLPermission != PackageManager.PERMISSION_GRANTED) {
            //用以申请权限的方法，此时使用 ActivityCompat 类的该方法，以便于版本兼容
            ActivityCompat.requestPermissions(this,
                    new String[]{Manifest.permission.READ_PHONE_STATE,
                            Manifest.permission.PROCESS_OUTGOING_CALLS},
                    1);
            return;
        } else {
            //如果已经获取了相关权限，调用 initData()与 initView()方法
            setCallReceiver();
        }
    } else {
        //如果 api 版本低于 23，直接调用 initData()与 initView()方法
        setCallReceiver();
    }
}

//申请权限做出响应后的回调函数
@Override
public void onRequestPermissionsResult(
        int requestCode, String[] permissions, int[] grantResults) {
    switch (requestCode) {
        case 1:
            if (grantResults[0] == PackageManager.PERMISSION_GRANTED) {
                Toast.makeText(this, "获取权限成功", Toast.LENGTH_SHORT)
                        .show();
                //获取权限成功，动态注册广播接收者
                setCallReceiver();
            } else {
                Toast.makeText(this, "获取权限失败", Toast.LENGTH_SHORT)
                        .show();
            }
```

```
                break;
            default:
                super.onRequestPermissionsResult(
                        requestCode, permissions, grantResults);
        }
    }
}
```

在 Activity 类中包括两部分内容：一部分是动态获取权限，一部分是动态注册广播。这里只讲解动态注册广播。首先我们创建一个 IntentFilter 实例，并为其添加一个值为 android.intent.action.PHONE_STATE 的 action。接下来创建一个 CallReceiver 实例，然后调用 registerReceiver()方法进行注册，将 CallReceiver 的实例和 IntentFilter 的实例都传进去，这样 CallReceiver 就会收到所有值为 android.intent.action.PHONE_STATE 的广播，也就实现了监听电话状态的功能。一般情况下，动态注册的广播接收器一定要取消注册才行，取消注册通过在 onDestroy()方法中通过调用 unregisterReceiver()方法来实现。

运行程序，如果你使用的手机系统是 6.0 以下版本，直接就获取了相关权限。但是如果你使用的手机系统是 6.0 及以上版本则会提示是否获取相关权限，点击“同意”即可，此时就会得到读取电话状态的权限。当获取到相关权限后，按 home 键使当前应用进入后台，打开拨号器，拨打一个电话，这时就会弹出一个 Toast，效果如图 9-1 所示。

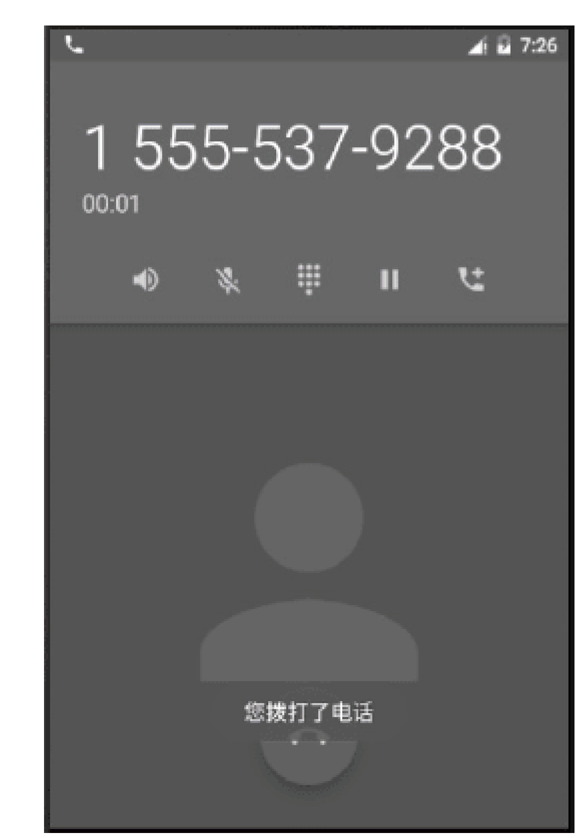


图 9-1　使用广播对拨打电话进行监听

这样就实现应用了对拨打电话的简单监听。但是在实际的开发实践中，用户通常更需要获得的是拨打的电话号码以及接听、挂断等状态的变化，甚至修改拨出的号码，比如加上一个 IP 拨号码。修改 CallBroadCast 类的 onReceive()方法如，代码如下：

```
@Override
public void onReceive(Context context, Intent intent) {
    boolean flag = false;
    //判断是来电还是去电
    if (intent.getAction().equals(Intent.ACTION_NEW_OUTGOING_CALL)) {
        // 标识当前是拨出电话
        flag = false;
        //获取拨出号码
        String phoneNumber = intent.
                getStringExtra(Intent.EXTRA_PHONE_NUMBER);
        Toast.makeText(context, "电话已拨出，号码为：" +
                phoneNumber, Toast.LENGTH_LONG).show();
        //将拨出号码改为代码 12159 的号码
        setResultData("12159"+phoneNumber);
    } else {
```

```
            //此时监控来电时状态
            //获取电话服务管理器 TelephonyManager
            TelephonyManager telephonyManager = (TelephonyManager)
                    context.getSystemService(Service.TELEPHONY_SERVICE);
            switch (telephonyManager.getCallState()) {
                //电话处于响铃状态
                case TelephonyManager.CALL_STATE_RINGING:
                    // 标识当前是来电
                    flag = true;
                    //获取来电号码
                    String incomingPhoneNumber =
                            intent.getStringExtra("incoming_number");
                    Toast.makeText(context, "来电号码：" +
                            incomingPhoneNumber, Toast.LENGTH_LONG).show();
                    break;
                case TelephonyManager.CALL_STATE_OFFHOOK:
                    if (flag) {
                        Toast.makeText(context, "来电已被接通",
                                Toast.LENGTH_LONG).show();
                    }
                    break;
                case TelephonyManager.CALL_STATE_IDLE:
                    if (flag) {
                        Toast.makeText(context, "来电已被挂断",
                                Toast.LENGTH_LONG).show();
                    }
                    break;
            }
        }
    }
```

在 onReceive()方法中，首先通过 intent.getAction()来判断是拨出电话还是外部来电。如果是拨出电话就获取拨出的号码，然后用 Toast 展示此号码，并使用 setResultData()方法将此号码修改为带 IP 的号码进行拨号。如果是来电，就先通过 getSystemService()方法获取 TelephonyManager 实例。这是一个系统服务类，专门用于管理通话。然后可以调用它的 getCallState()方法得到通话处于什么样状态，再根据不同的状态使用 Toast 展示不同的信息。

运行程序，按 home 键使当前程序进入后台，打开拨号界面，拨打电话，此时在界面上显示的号码就不再是拨出的号码，而是加上“12159”后的一个 IP 拨号号码。效果如图 9-2 所示。

这里只展示了拨出电话时的效果。有关来电时的效果，读者可以使用上例自行测试验证。

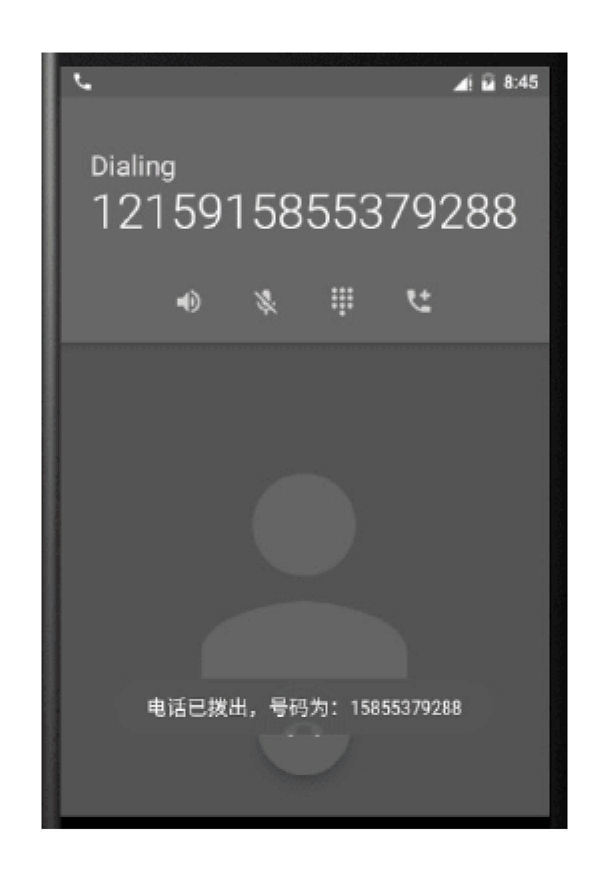


图 9-2　使用广播进行 IP 拨号

9.2.2　静态注册广播实例

动态注册的广播接收者可以自由地控制注册与注销，在灵活性方面有很大的优势，但是它也存在一个缺点，即必须在程序启动之后才能接收到广播，因为注册的逻辑是写在 onCreate()方法中的。但是，有时我们会需要在程序未启动的情况下就接收到广播，这时就需要使用静态注册的方式了。

下面用一个实例来讲解如何静态注册广播。本实例通过让程序接收一条开机广播（当收到这条广播时就在 onReceive()方法里执行相应的逻辑）来实现开机启动的功能。先新建一个广播接收者类 SteadyReceiver，代码如下：

```
public class SteadyReceiver extends BroadcastReceiver{
    @Override
    public void onReceive(Context context, Intent intent) {
        Toast.makeText(context,"我已经被开启了",Toast.LENGTH_LONG).show();
    }
}
```

此时并不需要在 Activity 中进行操作，只需要在 AndroidManifest.xml 文件中与<Activity>标签并列的位置加入如下代码即可：

```
<receiver android:name=".broadcast.SteadyReceiver">
            <intent-filter>
                <action android:name="android.intent.action.BOOT_COMPLETED" />
            </intent-filter>
    </receiver>
```

静态注册广播的用法其实和 Service 的注册非常相似，首先通过 android:name 来指定具体注册哪一个广播接收器，然后在<intent-filter>标签里加入想要接收的广播就行了。由于 Android 系统启动完成后会发出一条值为 android.intent.action.BOOT_COMPLETED 的广播，因此我们在这里添加了相应的 action。

另外，监听系统开机广播需要权限，但是此权限属于普通权限，只需要在 AndroidManifest.xml 中加入如下权限即可：

```
<uses-permission
android:name="android.permission.RECEIVE_BOOT_COMPLETED"/
>
```

此时将程序安装进模拟器，然后重新启动模拟器，当模拟器被打开时就会有 Toast 弹出，说明此应用已经被开启，效果如图 9-3 所示。

通过本节的学习，读者应该能够掌握如何使用广播接收者来接收系统广播的内容，并通过 onReceive()方法处理相关逻辑，下一节我们将讲解如何自定义广播。



图 9-3 使用广播实现开机启动

9.3 自定义广播：普通广播与有序广播

通过 9.2 节的学习，我们应该已经学会了通过广播接收者来接收系统广播的内容，但是在实际开发中，仍需要自定义一些广播。本节我们就来讲解如何在应用程序中发送自定义的广播。在 9.1 节就已经指出 BroadcastReceiver 所对应的广播分两类，即普通广播和有序广播。其中，普通广播通过 Context.sendBroadcast()方法来发送，完全异步，广播接收者的执行顺序不确定、效率高，但是无法使用 setResult()等方法来设置广播传递的值，也无法中断广播；有序广播通过 Context.sendOrderedBroadcast()来发送，所有的 receiver 依次执行。BroadcastReceiver 可以使用 setResult()等方法来设置结果并传给下一个 BroadcastReceiver，通过 getResult()等方法来取得上一个广播接收者返回的结果，并可以用 abort()等方法来让系统丢弃该广播，使该广播不再传送到别的广播接收者。

下面通过实例来讲解如何使用这两种广播。

9.3.1 普通广播实例

发送广播很简单，只需要声明一个意图，然后使用 Context.sendBroadcast()方法发送意图即可。这里在布局文件中加入一个 Button 按钮来触发发送广播的事件，activity_custom_broadcast.xml 文件代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.buaa.broadcast.activity.CustomBroadcastActivity">

    <Button
```

```
            android:id="@+id/send"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="发送广播" />
</RelativeLayout>
```

在 Activity 中对按钮的点击事件进行处理，发送广播。CustomBroadcastActivity 类的代码如下：

```
public class CustomBroadcastActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_custom_broadcast);

        findViewById(R.id.send).setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                //建立一个意图，action 为 com.buaa.braodcast.NORMAL_BROADCAS
                Intent intent = new Intent("com.buaa.action.NORMAL_BROADCAST");
                Bundle bundle = new Bundle();
                bundle.putString("name", "李瑞奇");
                bundle.putString("job", "工程师");
                //向意图中加入数据
                intent.putExtra("data", bundle);
                //发送广播，普通广播
                sendBroadcast(intent);
            }
        });
    }
}
```

这里的代码很简单，只是在按钮的点击事件里面加入了发送自定义广播的逻辑。首先构建出一个 Intent 对象，并使用 Bundle 存储要传递的值，并用 intent.putExtra("data", bundle)把要发送的值传入，然后调用了 Context 的 sendBroadcast()方法将广播发送出去，这样所有监听 com.buaa.action.NORMAL_BROADCAST 这条广播的广播接收者就会收到消息。普通广播被发送之后，还需要定义一个广播接收者来准备接收此广播。这里新建一个 CustomReceiver 类继承自 BroadcastReceiver 类，代码如下：

```
package com.buaa.broadcast.broadcast;

import android.content.BroadcastReceiver;
import android.content.Context;
```

```
import android.content.Intent;
import android.os.Bundle;
import android.widget.Toast;

public class CustomReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        //获取广播的 action
        String action = intent.getAction();
        //获取广播传递的数据
        Bundle bundle = intent.getBundleExtra("data");
        String name = (String) bundle.get("name");
        String job = (String) bundle.get("job");
        Toast.makeText(context,
                "接收到自定义的普通广播,其中 action 为："+action+
                        "；接收的广播数据为，姓名："+name+",工作："+job,
                Toast.LENGTH_LONG).show();
    }
}
```

逻辑很简单，当 CustomReceiver 收到自定义的广播时获取相关的数据，并弹出“接收到自定义的普通广播”和接收到的数据。除了上述代码外，还需要在 AndroidManifest.xml 中对这个广播接收者进行注册：

```
<receiver android:name=".broadcast.CustomReceiver">
    <intent-filter>
        <action android:name="com.buaa.action.NORMAL_BROADCAST" />
    </intent-filter>
</receiver>
```

<intent-filter>标签中 action 的 name 值正是在 Activity 类中声明的 Intent 的 action，两者需要保持一致。运行程序并点击按钮，此时将发出一条普通广播，如图 9-4 所示。



图 9-4　接收自定义广播的信息

这样就成功完成了发送自定义普通广播的功能，需要说明的是这里的广播在其他应用中也是可以接收到的。读者可以新建一个应用，使用同样的接收者类即可，然后观察是否能够接收到相应的广播。

9.3.2　有序广播实例

和发送普通广播相比，发送有序广播只需要改动一行代码，即将 sendBroadcast()方法改成 sendOrderedBroadcast()方法。sendOrderedBroadcast()方法接收两个参数，第一个参数仍然是 Intent，第二个参数是一个与权限相关的字符串，这里传入 null

就行了。Activity 类中的 onCreate()代码如下：

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_custom_broadcast);
    findViewById(R.id.send).setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            //建立一个意图，action 为 com.buaa.action.NORMAL_BROADCAS
            Intent intent = new Intent("com.buaa.action.NORMAL_BROADCAST");
            Bundle bundle = new Bundle();
            bundle.putString("name", "李瑞奇");
            bundle.putString("job", "工程师");
            //向意图中加入数据
            intent.putExtra("data", bundle);
            //发送广播，普通广播
            sendOrderedBroadcast(intent, null);
        }
    });
}
```

运行程序，会发现效果与普通广播并无区别。这个时候的广播接收者是有先后顺序的，而且前面的广播接收者还可以将广播截断，以阻止其继续传播。决定接收先后顺序的是广播接收者的优先级，优先级的范围为-1000 到 1000，默认为 0，优先级越大越要优先执行。为了让读者能够更直观地感受有序广播的不同，下面新建一个应用 OrderReceiver，并在这个应用中使用广播接收者接收 com.buaa.action.NORMAL_BROADCAST 这个广播。广播接收者代码如下：

```
public class CustomReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        //获取传递的参数
        Bundle bundle = this.getResultExtras(true);
        String name = (String) bundle.get("name");
        String job = (String) bundle.get("job");
        Toast.makeText(context,
                "在 OrderReceiver 应用中接收到广播；接收的广播数据为，姓名："
                        + name + ",工作：" + job,
                Toast.LENGTH_LONG).show();
    }
}
```

然后在 AndroidManifest.xml 中进行注册，代码如下：

```
<receiver android:name="com.buaa.orderreceiver.CustomReceiver">
    <intent-filter android:priority="50">
        <action android:name="com.buaa.action.NORMAL_BROADCAST" />
    </intent-filter>
</receiver>
```

这样就完成了在 OrderReceiver 应用中使用广播接收者接收广播的开发，接着安装此应用。为了凸显不同优先级的作用，需要修改 broadcast 应用中广播接收者的代码：

```
public class CustomReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        //获取传递的参数
        Bundle bundle = intent.getBundleExtra("data");
        String name = bundle.getString("name");
        String job = bundle.getString("job");
        bundle.putString("name", "莫言");
        bundle.putString("job", "作家");
        //修改传递的参数，下一个接收者获取的将是修改过的参数
        this.setResultExtras(bundle);
        Toast.makeText(context,
                "在 broadcast 应用中接收到广播；" +
                        "接收的广播数据为，姓名：" + name + ",工作：" + job,
                Toast.LENGTH_LONG).show();
    }
}
```

并在 AndroidManifest.xml 文件中修改优先级：

```
<receiver android:name=".broadcast.CustomReceiver">
    <intent-filter android:priority="100">
        <action android:name="com.buaa.action.NORMAL_BROADCAST" />
    </intent-filter>
</receiver>
```

重新运行程序，点击“发送广播”按钮，将会出现两次 Toast 弹窗：第一次是 broadcast 应用的广播接收者接收到广播，如图 9-5 所示；第二次是 OrderReceiver 应用中的广播接收者接收到的经过 broadcast 应用的广播接收者处理之后的内容，如图 9-6 所示。



图 9-5　有序广播：在 broadcast 应用中接收广播



图 9-6　有序广播：在 OrderReceiver 应用中接收广播

在这个实例中，我们可以看到优先级相对较高的 broadcast 应用中的广播接收者先接收到了广播，并对广播传递的数据进行了处理。除了这些操作之外，现接收到广播的接收者还可以中断广播，方法很简单，只需要在 onReceive()方法中调用 abortBroadcast()方法即可，修改如下：

```
@Override
public void onReceive(Context context, Intent intent) {
    //获取传递的参数
    Bundle bundle = intent.getBundleExtra("data");
    String name = bundle.getString("name");
    String job = bundle.getString("job");

    bundle.putString("name", "莫言");
    bundle.putString("job", "作家");
    //修改传递的参数，下一个接收者获取的将是修改过的参数
    this.setResultExtras(bundle);
    Toast.makeText(context,
            "在 broadcast 应用中接收到广播；" +
                    "接收的广播数据为，姓名：" + name + ",工作：" + job,
            Toast.LENGTH_LONG).show();
    abortBroadcast();
}
```

如果此时重写运行程序，就会发现只有 braodcast 中的接收者能够接收到广播并弹出 Toast，说明这条广播确实被中断了。

9.4 使用本地广播

前面我们发送和接收的广播全部属于系统全局广播，即发出的广播可以被其他任何应用程序接收到，并且我们也可以接收来自于其他任何应用程序的广播。这样就很容易会引起安全性的问题，比如说我们发送的一些携带关键性数据的广播有可能被其他的应用程序截获或者其他的程序不停地向我们的广播接收器里发送各种垃圾广播。

为了能够简单地解决广播的安全性问题，Android 引入了一套本地广播机制，使用这个机制发出的广播只能够在应用程序的内部进行传递，并且广播接收器也只能接收来自本应用程序发出的广播，这样所有的安全性问题就都不存在了。另外，发送本地广播比起发送系统全局广播效率更高。

本地广播的用法并不复杂，主要就是使用了一个 LocalBroadcastManager 来对广播进行管理，并提供了发送广播和注册广播接收器的方法。下面我们就通过具体的实例来演示它的用法。直接在 9.3 节的 broadcast 应用中进行修改。由于是在应用内进行广播，因此 CustomReceiver 的 onReceive()方法中并不需要对广播传递的内容进行修改，既不需要同时也不需要在终端广播，将这两部分删除，修改代码如下：

```
public class CustomReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        //获取传递的参数
        Bundle bundle = intent.getBundleExtra("data");
        String name = bundle.getString("name");
        String job = bundle.getString("job");

        Toast.makeText(context,
                "在 broadcast 应用中接收到广播；" +
                        "接收的广播数据为，姓名：" + name + ",工作：" + job,
                Toast.LENGTH_LONG).show();
    }
}
```

在 CustomBroadcastActivity 类中要做较大的改动，修改如下：

```
public class CustomBroadcastActivity extends AppCompatActivity {

    private LocalBroadcastManager localBroadcastManager;
    private CustomReceiver customReceiver;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
```

```
        setContentView(R.layout.activity_custom_broadcast);
        localBroadcastManager = LocalBroadcastManager.getInstance(this);

        findViewById(R.id.send).setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                //建立一个意图，action 为 com.buaa.action.NORMAL_BROADCAS
                Intent intent = new Intent("com.buaa.action.NORMAL_BROADCAST");
                Bundle bundle = new Bundle();
                bundle.putString("name", "李瑞奇");
                bundle.putString("job", "工程师");
                //向意图中加入数据
                intent.putExtra("data", bundle);
                //发送广播，普通广播
                localBroadcastManager.sendBroadcast(intent);
            }
        });

        IntentFilter intentFilter = new IntentFilter();
        intentFilter.addAction("com.buaa.action.NORMAL_BROADCAST");
        customReceiver = new CustomReceiver();
        localBroadcastManager.registerReceiver(customReceiver, intentFilter);
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
        localBroadcastManager.unregisterReceiver(customReceiver);
    }
}
```

这部分代码和我们前面所学的动态注册广播接收者以及发送广播的代码是一样的。只不过现在首先是通过 LocalBroadcastManager 的静态方法 getInstance()得到了 LocalBroadcast-Manager 的一个实例，然后用 LocalBroadcastManager 对象调用 registerReceiver()方法注册广播接收者，再用 LocalBroadcastManager 对象调用 endBroadcast()方法发送广播。

本地广播是无法通过静态注册的方式来接收的。其实这也完全可以理解，因为静态注册主要就是为了让程序在未启动的情况下也能收到广播，而发送本地广播时，我们的程序肯定是已经启动了，完全不需要使用静态注册的功能，所以也应将 AndroidManifest.xml 文件中的静态注册接收者部分删除。

重新运行程序并点击按钮，效果如图 9-7 所示。

此时试图让 OrderReceiver 应用接收 com.buaa.action.NORMAL_BROADCAST 这条广播，将会无法接收到。



图 9-7　接收本地广播的内容

9.5 小　　结

本章系统讲解了广播机制，并通过实例告诉读者如何使用系统广播，以及通过对普通广播和有序广播的介绍讲解了如何自定义广播。本章的最后讲解了本地广播，这是 Android 为了能够简单地解决广播的安全性问题而引入的一套本地广播机制。

至此，Android 的四大组件（Activity、Service、ContentProvider 和 Broadcast）已经全部讲完。Broadcast 是四大组件之一，可见其重要性，希望读者能够加以重视。



第10章 网络开发

如果让笔者来划分时代，那么笔者会说农业时代、工业时代、信息时代和移动互联网时代。不要提互联网时代，互联网还算不上一个时代，只是信息时代的一个部分，笔者想以此来证明移动互联网有多重要。农业时代主要改变了生产关系，人们摆脱了对狩猎的依赖。工业时代则是对能量资源的大肆开发和利用。工业时代后期我们进入了电气时代，而计算机的发明使我们进入了信息时代。这时的表现为信息量、信息传播、信息处理的速度等都呈几何倍增，乃至形成信息爆炸。移动互联网时代则完成了最后一步，让人与信息相连，还记得阿凡达么，里面的每个人身上都长了一个接口，可以随时和星球乃至动物相连，互传信息，是的，移动终端就是这么一个接口。

这是笔者曾经读过的一篇文章，至今记忆深刻。移动互联网时代的载体是移动终端，依靠的技术就是网络通信技术。本章就来讲解 Android 中网络通信与常见的开发技术。

10.1 Android 网络通信概述

Android 常用的网络通信技术主要有两种，一种是使用 HTTP 协议进行网络通信，另一种是用 Socket 进行网络通信。而网络通信这两种方式都离不开 TCP/IP 网络协议。

10.1.1 TCP/IP

TCP/IP（Transmission Control Protocol/Internet Protocol，传输控制协议/网间网协议）是目前世界上应用最为广泛的协议，它的流行与 Internet 的迅猛发展密切相关——TCP/IP 最初是为互联网的原型 ARPANET 所设计的，目的是提供一整套方便实用、能应用于多种网络上的协议，事实证明 TCP/IP 做到了这一点，它使网络互联变得容易起来，并且使越来越多的网络加入其中，成为 Internet 的事实标准。TCP/IP 协议族包含了很多功能各异的子协议，如果按照分层的方式来剖析结构，那么 TCP/IP 层次模型共分为 4 层：应用层、传输层、网络层、数据链路层。

（1）应用层——所有用户所面向的应用程序的统称。ICP/IP 协议族在这一层面有很多协议来支持不同的应用，许多大家所熟悉的基于 Internet 的应用实现就离不开这些协议。例如，我们进行万维网（WWW）访问用到的 HTTP、文件传输用的 FTP、电子邮件发送用的 SMTP、域名的解析用的 DNS 协议、远程登录用的 Telnet 协议等，都是属于 TCP/IP 应用层的；就用户而言，看到的是由一个个软件所构筑的大多为图形化的操作界面，而实际后台运行的便是上述协议。

（2）传输层——主要功能是提供应用程序间的通信，TCP/IP 协议族在这一层的协议有 TCP 和 UDP。

（3）网络层——TCP/IP 协议族中非常关键的一层，主要定义了 IP 地址格式，从而能够使得不同应用类型的数据在 Internet 上通畅地传输。IP 协议就是一个网络层协议。

（4）网络接口层——TCP/IP 软件最低层，负责接收 IP 数据包并通过网络发送，或者从网络上接收物理帧，抽出 IP 数据报，交给 IP 层。

10.1.2 HTTP 与 Socket

HTTP 即超文本传送协议，是 Web 联网的基础，也是手机联网常用的协议之一，处于 TCP/IP 协议中的应用层。 HTTP 连接最显著的特点是客户端发送的每次请求都需要服务器回送响应，在请求结束后会主动释放连接。从建立连接到关闭连接的过程称为“一次连接”。由于 HTTP 在每次请求结束后都会主动释放连接，因此 HTTP 连接是一种“短连接”。

Socket（套接字）则是对 TCP/IP 的封装和应用。Socket 本身并不是协议，而是一个 API，通过 Socket 我们可以使用 TCP/IP。实际上，Socket 跟 TCP/IP 没有必然的联系。Socket 编程接口在设计的时候就希望也能适应其他的网络协议。所以说，Socket 的出现只是使得程序员更方便地使用 TCP/IP 协议栈而已，是对 TCP/IP 协议的抽象，从而形成了我们知道的一些最基本的函数接口，比如 create、listen、connect、accept、send、read 和 write 等。由于通常情况下

Socket 连接就是 TCP 连接，因此 Socket 连接一旦建立，通信双方就可开始相互发送数据内容，直到双方连接断开。但在实际网络应用中，客户端到服务器之间的通信往往需要穿越多个中间节点，例如路由器、网关、防火墙等，大部分防火墙默认会关闭长时间处于非活跃状态的连接而导致 Socket 连接断开，因此需要通过轮询告诉网络该连接处于活跃状态。

有些情况下，需要服务器端主动向客户端推送数据，保持客户端与服务器数据的实时与同步。此时若双方建立的是 Socket 连接，服务器就可以直接将数据传送给客户端；若双方建立的是 HTTP 连接，则服务器需要等到客户端发送一次请求后才能将数据传回给客户端，因此，客户端定时向服务器端发送连接请求，不仅可以保持在线，同时也是在“询问”服务器是否有新的数据，如果有就将数据传给客户端。

通过对两者的分析，可以总结出它们的优缺点以及适用场景（见表 10-1）。

表10-1　HTTP与Socket的对比

	使用 Socket 进行网络通信	使用 HTTP 进行网络通信
优点	① 传输数据为字节级，传输数据可自定义，数据量小，传输时间短，性能高 ② 适合于客户端和服务器端之间信息实时交互 ③ 可以加密，数据安全性强	① 基于应用级的接口，使用方便 ② 程序员开发水平要求不高，容错性强
缺点	① 需对传输的数据进行解析，转化成应用级的数据 ② 对开发人员的开发水平要求高 ③ 相对于 Http 协议传输，增加了开发量	① 传输速度慢，数据包大（HTTP 中包含辅助应用信息） ② 数据传输安全性差
应用场景	适合于对传输速度、安全性、实时交互、费用等要求高的应用中，如网络游戏、手机应用、银行内部交互、即时聊天等	适合于对传输速度、安全性要求不是很高且需要快速开发的应用

关于 TCP/IP、HTTP、Socket，这里就不过多介绍了，如果读者想要了解，可以阅读专门的网络编程书籍。后面我们将详细讲解在 Android 中如何使用 HTTP 以及 Socket 进行网络通信。

10.2　使用 HTTP 协议进行网络通信

在 Android 中使用 HTTP 进行网络通信的方式一般有两种，即 Android 自带的 HttpURLConnection 以及第三方开源的网络通信框架。这里说的第三方开源的网络通信框架有很多，现在比较常用的是 OkHttp。

有的 Android 开发书籍中重点讲解的 HttpClient 是 Apache 公司的一款开源框，可以很好地支持很多细节的控制（如代理、COOKIE、鉴权、压缩、连接池），所以曾经使用者较多，甚至在早期的 Android 版本中，SDK 是直接集成 HttpClient 的。由于对开发人员要求相对较高，代码写起来更复杂，很难被普通开发人员很好地驾驭，官方对 HttpClient 的支持也越来越少，现

在几乎无人使用了，所以这里就不讲解 HttpClient 了。本节我们将主要讲解 HttpURLConnection 的用法。

10.2.1 HttpURLConnection 简介

HttpURLConnection 是 Android 官方支持的网络通信接口，直接支持系统级连接池，即打开的连接不会直接关闭，在一段时间内所有程序可共用；直接在系统层面做了缓存策略处理，加快重复请求的速度；直接支持 GZIP 压缩。使用 HttpURLConnection 首先需要了解如下几个常用类和它们的常用方法。

（1）URL 类

URL 类主要的功能是定位到要获取资源的网址以及打开连接，比如下面的代码：

```
URL url = new URL("192.168.1.104:8080");
HttpURLConnection connection = (HttpURLConnection) url.openConnection();
```

（2）HttpURLConnection 类

此类是通信的核心类，连接设置都需要通过该类。这里需要使用到很多方法，比如设置请求方式为 POST，设置需要发送过去的数据以及设置超时时间，获得返回的数据。HttpURLConnection 类的常用方法如表 10-2 所示。

表10-2　HttpURLConnection类常用方法

方法	作用
setDoOutput(Boolean b)	设置是否可以写入数据
setRequestMethod(String str)	设置请求的方式（"GET"、"POST"）
getOutputStream()	获得输出流对象，通过此方法可以向请求中写数据
getInputStream()	获得输入流对象，通过此方法获取网站返回的数据
setConnectTimeout(int time)	设置超时时间
disconnect()	关闭当前的 HTTP 连接

10.2.2 HttpURLConnection 使用实例

下面通过实例来分别讲解如何使用 GET 请求和 POST 请求。新建一个项目 http，然后创建一个处理 HTTP 请求的工具类 HttpUtil，代码如下：

```
package com.buaa.http;

import android.util.Log;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.HttpURLConnection;
import java.net.URL;
```

```
import java.util.Map;

public class HttpUtil {
    public static String get(String ip) {
        String result = "";
        HttpURLConnection conn = null;
        BufferedReader in = null;
        try {
            URL url = new URL(ip);
            //得到 HttpURLConnection 实例化对象
            conn = (HttpURLConnection) url.openConnection();
            //设置请求方式
            conn.setRequestMethod("GET");
            //conn.setRequestProperty("encoding","UTF-8"); //可以指定编码
            //设置请求方式和响应时间
            conn.setConnectTimeout(5000);
            //不使用缓存
            conn.setUseCaches(false);
            //读取响应
            if (conn.getResponseCode() == 200) {
                in = new BufferedReader(
                        new InputStreamReader(conn.getInputStream()));
                String line;
                while ((line = in.readLine()) != null) {
                    result += "/n" + line;
                }
            } else {
                Log.i("connect", "请求失败");
            }
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            //释放资源
            if (in != null) {
                try {
                    in.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
            if (conn != null) {
                conn.disconnect();
            }
```

```
        }
        return result;
    }

    public static String post(String url, Map<String, String> map) {
        PrintWriter out = null;
        BufferedReader in = null;
        String result = "";
        HttpURLConnection conn = null;
        try {
            URL realUrl = new URL(url);
            // 打开和 URL 之间的连接
            conn = (HttpURLConnection) realUrl.openConnection();
            // 设置通用的请求属性
            conn.setRequestProperty("accept", "*/*");
            conn.setRequestProperty("connection", "Keep-Alive");
            conn.setRequestProperty("user-agent",
                    "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1    ; SV1)");
            // 发送 POST 请求必须设置如下两行
            conn.setDoOutput(true);
            conn.setDoInput(true);
            // 获取 URLConnection 对象对应的输出流
            out = new PrintWriter(conn.getOutputStream());
            String data = "";
            for (Map.Entry<String, String> entry : map.entrySet()) {
                data += entry.getKey() + "=" + entry.getValue() + "&";
            }
            // 发送请求参数
            out.print(data);
            // flush 输出流的缓冲
            out.flush();
            // 定义 BufferedReader 输入流来读取 URL 的响应
            in = new BufferedReader(
                    new InputStreamReader(conn.getInputStream()));
            String line;
            while ((line = in.readLine()) != null) {
                result += line;
            }
        } catch (Exception e) {
            System.out.println("发送 POST 请求出现异常！ " + e);
            e.printStackTrace();
        }
        // 使用 finally 块来关闭输出流、输入流
```

```
        finally {
            try {
                if (out != null) {
                    out.close();
                }
                if (in != null) {
                    in.close();
                }
            } catch (IOException ex) {
                ex.printStackTrace();
            }
            if (conn != null) {
                conn.disconnect();
            }
        }
        return result;
    }
}
```

为了方便读者阅读，代码中加入了大量注释，这样一来代码应该很容易理解。在 HttpUtil 类中我们创建了两个方法，分别用来处理 GET 请求和 POST 请求。需要注意的是，在发送请求时，POST 请求是通过 PrintWriter 类将输出流进行包装再用 PrintWriter 类的 print()方法发送数据的，被发送的数据都要使用键值对的形式，多个数据时，中间用“&”隔开。但是 GET 请求是不能通过这种方式的。如果 GET 请求想要传递参数，只能在链接的最后加上“?Param1=value1&param2=value2”的形式，比如“192.168.1.104:8080?Name=ricky& psd=123”。HttpUtil 类中的两个方法都返回了 String 类型的返回值，这将在 MainActivity 中被使用。

为了方便进行网络通信，在布局文件中我们使用两个 EditText 输入数据、两个 Button 用于触发 GET 与 POST 请求，同时使用了一个新的控件 ScrollView。由于手机屏幕的空间一般都比较小，有时过多的内容一屏是显示不下的，借助 ScrollView 控件就可以允许我们以滚动的形式查看屏幕外的那部分内容。ScrollView 标签内通常可以包含一个且只能是一个控件，即 TextView。布局文件 activity_main.xml 的代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context="com.buaa.http.MainActivity">

    <EditText
        android:id="@+id/name"
        android:layout_width="match_parent"
```

```
            android:layout_height="wrap_content"
            android:hint="请输入用户名："
            android:textSize="24sp" />

        <EditText
            android:id="@+id/psd"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:hint="请输入密码："
            android:inputType="textPassword"
            android:textSize="24sp" />

        <Button
            android:id="@+id/post"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="点击登录（POST）" />

        <Button
            android:id="@+id/get"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="获取文章（GET）" />

        <ScrollView
            android:layout_width="match_parent"
            android:layout_height="wrap_content">

            <TextView
                android:id="@+id/data"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:textSize="24sp" />
        </ScrollView>
</LinearLayout>
```

接着在 MainActivity 中通过 findViewById()方法获取布局中的各个控件，然后通过按钮触发网络请求，POST 请求发送 EditText 中的数据，GET 请求则从服务端获取数据。又由于在 Android 中网络请求的操作不能在主线程中进行，因此必须开启子线程来进行网络请求，代码如下：

```
package com.buaa.http;

import android.os.Handler;
```

```
import android.os.Message;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;
import android.widget.Toast;

import java.util.HashMap;
import java.util.Map;

public class MainActivity extends AppCompatActivity implements View.OnClickListener {

    private TextView textView;
    private Button getButton;
    private Button postButton;
    private EditText psdEditText;
    private EditText nameEditText;

    final int GET = 123;
    final int POST = 124;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        initView();
    }

    private void initView() {
        textView = (TextView) findViewById(R.id.data);
        getButton = (Button) findViewById(R.id.get);
        postButton = (Button) findViewById(R.id.post);
        getButton.setOnClickListener(this);
        postButton.setOnClickListener(this);

        nameEditText = (EditText) findViewById(R.id.name);
        psdEditText = (EditText) findViewById(R.id.psd);
    }

    @Override
    public void onClick(View v) {
```

```
        switch (v.getId()) {
            case R.id.get:
                //必须使用子线程
                new Thread(new Runnable() {
                    @Override
                    public void run() {
                        //这个 IP 地址是笔者的 PC 在局域网中的 IP 地址，端口号是服务端的端口号
                        String result = HttpUtil.get("http://192.168.1.104:8080?name=ricky");
                        Message msg = handler.obtainMessage();
                        msg.what = GET;
                        msg.obj = result;
                        handler.sendMessage(msg);
                    }
                }).start();
                break;
            case R.id.post:
                //必须使用子线程
                new Thread(new Runnable() {
                    @Override
                    public void run() {
                        Map<String, String> map = new HashMap();
                        map.put("name", nameEditText.getText().toString());
                        map.put("psd", psdEditText.getText().toString());
                        String result = HttpUtil.post("http://192.168.1.104:8080", map);
                        Message msg = handler.obtainMessage();
                        msg.what = POST;
                        msg.obj = result;
                        handler.sendMessage(msg);
                    }
                }).start();
                break;
        }
    }

    private Handler handler = new Handler() {
        @Override
        public void handleMessage(Message msg) {
            super.handleMessage(msg);
            switch (msg.what) {
                case GET:
                    String text = (String) msg.obj;
                    textView.setText(text);
                    break;
```

```
                    case POST:
                        Toast.makeText(MainActivity.this, (String) msg.obj, Toast.LENGTH_LONG).show();
                        break;
                }
            }
        };
    }
```

在 MainActivity 中，当触发完网络请求之后，GET 请求返回的数据展示到 TextView 上，POST 请求返回的数据用 Toast 展示。由于子线程不能修改 UI 界面，因此修改 TextView 必须使用 Handler 机制。在子线程内创建一个 Message 对象，并使用 Handler 将它发送出去。之后又在 Handler 的 handleMessage()方法中对这条 Message 进行处理，将结果设置到 TextView 上展现出来。

在 Android 系统中，使用网络需要申请权限。由于网络权限是普通权限，因此只需要在 AndroidManifest.xml 文件中进行静态申请即可。具体来说就是添加如下代码：

```
<uses-permission android:name="android.permission.INTERNET" />
```

完成了权限申请就可以运行程序了。为了保证网络通信的准确，这里使用真机进行演示。本实例中访问的服务器是笔者使用 JavaEE 和 Tomcat 搭建的，与手机处于同一个局域网。开启 WiFi 之后，运行程序。

在两个 EditText 中分别输入“ricky”和“123”，点击“点击登录（POST）”时将能够验证正确，Toast 展示出“成功登录”，如图 10-1 所示，而当输入的内容不正确时，Toast 则会展示出“用户名不正确”或者“密码不正确”，分别如图 10-2 与图 10-3 所示。此时如果点击“获取文章（GET）”，从服务端读取的文章将会展现在 TextView 中，由于使用 ScrollView，因此文章内容可以上下滑动翻看，如图 10-4 所示。



图 10-1　使用 HttpURLConnection 类进行网络通信：登录成功



图 10-2　使用 HttpURLConnection 类进行网络通信：用户名不存在



图 10-3 使用 HttpURLConnection 类进行网络通信：密码不正确



图 10-4 使用 HttpURLConnection 类进行网络通信：获取文章

10.3 客户端类库 OkHttp

在学习了如何使用HttpURLConnection之后，读者应该已经能够处理基本的HTTP请求了。但是 HttpURLConnection 在使用上有诸多不便，为此，Square 公司实现了一个 HTTP 客户端的类库——OkHttp。这里不得不提一下，在较新的 Android SDK 中，HttpURLConnection 的底层就是使用 OkHttp 来实现的，当然 Google 公司也对此做了一些优化，这里可见 OkHttp 是多么的成功，这也是我们需要重点学习它的一个原因。

10.3.1 OkHttp 简介

Okhttp 是一个支持 HTTP 和 HTTP/2 的客户端，可以在 Android 和 Java 应用程序中使用，具有以下特点：

（1）API 设计轻巧，基本上通过几行代码的链式调用即可获取结果。

（2）既支持同步请求，也支持异步请求。同步请求会阻塞当前线程，异步请求不会阻塞当前线程，异步执行完成后执行相应的回调方法。

（3）其支持 HTTP/2 协议，通过 HTTP/2，可以让客户端中到同一服务器的所有请求共用同一个 Socket 连接。

（4）如果请求不支持 HTTP/2 协议，那么 OkHttp 会在内部维护一个连接池，通过该连接池，可以对 HTTP/1.x 的连接进行重用，减少了延迟。

（5）透明的 GZIP 处理降低了下载数据的大小。

（6）请求的数据会进行相应的缓存处理，下次再进行请求时，如果服务器告知 304（表明数据没有发生变化），就直接从缓存中读取数据，降低了重复请求的数量。

在使用 OkHttp 之前，我们先来介绍几个核心类：OkHttpClient、Request、Call 和 Response。

1. OkHttpClient

OkHttpClient 表示了 HTTP 请求的客户端类，在绝大多数的 App 中，我们只应该执行一次 new OkHttpClient()，将其作为全局的实例进行保存，从而在 App 的各处都只使用这一个实例对象，这样所有的 HTTP 请求都可以共用 Response 缓存、共用线程池以及连接池。

默认情况下，直接执行 OkHttpClient client = new OkHttpClient()就可以实例化一个 OkHttpClient 对象。但是，当我们需要配置 OkHttpClient 的一些参数（比如超时时间、缓存目录、代理、Authenticator 等）时，就要用到内部类 OkHttpClient.Builder 了，设置如下：

```
OkHttpClient client = new OkHttpClient.Builder().
        readTimeout(30, TimeUnit.SECONDS).
        cache(cache).
        proxy(proxy).
        authenticator(authenticator).
        build();
```

OkHttpClient 本身不能设置参数，需要借助于其内部类 Builder 设置参数，参数设置完成后，调用 Builder 的 build 方法得到一个配置好参数的 OkHttpClient 对象。这些配置的参数会对该 OkHttpClient 对象所生成的所有 HTTP 请求都有影响。有时候我们想单独给某个网络请求设置特别的几个参数，比如只想让某个请求的超时时间设置为一分钟，但是还想保持 OkHttpClient 对象中的其他参数设置，就可以调用 OkHttpClient 对象的 newBuilder()方法，代码如下：

```
OkHttpClient client2 = client.newBuilder().
        readTimeout(60, TimeUnit.SECONDS).
        build();
```

此时，这个 client2 的配置还是使用 client 的配置，只不过是覆盖了超时时间这一项。

2. Request

Request 类封装了请求报文信息：请求的 Url 地址、请求的方法（如 GET、POST 等）、各种请求头（如 Content-Type、Cookie）以及可选的请求体。一般通过内部类 Request.Builder 的链式调用生成 Request 对象。

3. Call

Call 代表了一个实际的 HTTP 请求，是连接 Request 和 Response 的桥梁，通过 Request 对象的 newCall()方法可以得到一个 Call 对象。Call 对象既支持同步获取数据，也可以异步获取数据。

执行 Call 对象的 execute()方法，会阻塞当前线程去获取数据，该方法返回一个 Response 对象。

执行 Call 对象的 enqueue()方法，不会阻塞当前线程，该方法接收一个 Callback 对象，当异步获取到数据之后，会回调执行 Callback 对象的相应方法。如果请求成功，就执行 Callback

对象的 onResponse 方法，并将 Response 对象传入该方法中；如果请求失败，就执行 Callback 对象的 onFailure 方法。

4. Response

Response 类封装了响应报文信息：状态吗（200、404 等）、响应头（Content-Type、Server 等）以及可选的响应体。可以通过 Call 对象的 execute()方法获得 Response 对象，异步回调执行 Callback 对象的 onResponse 方法时也可以获取 Response 对象。

介绍了 OkHttp 的基本状况之后，下面就来学习如何使用 OkHttp。当然如果想要在 AndroidStudio 中使用 OkHttp，还需要在 build.gradle（app 中的 build.gradle）中加入引入 jar 包的命令：

```
compile 'com.squareup.okhttp3:okhttp:3.3.1'
```

10.3.2 OkHttp 中各种请求的实现

OkHttp 是当下最流行、使用最广泛的 Http 请求框架，本节将以代码的方式详细讲解如何处理在实际开发中会使用到的各种请求。

1. 同步 GET 请求

使用同步 GET 请求的步骤是：创建一个的 OkHttpClient 对象，对象引用为 client；根据传入的 url 参数构建请求报文 Request；使用 client 执行 newCall()方法会得到一个 Call 对象，表示一个新的网络请求，此处 newCall 方法的参数就是上一步创建的 Request；Call 对象的 execute()方法是同步方法，会阻塞当前线程，返回 Response 对象；通过 Response 对象的 isSuccessful()方法可以判断请求是否成功；通过 Response 对象的 body()方法可以得到响应体 ResponseBody 对象，调用其 string()方法可以很方便地将响应体中的数据转换为字符串，该方法会将所有的数据放入内存中，所以如果数据超过 1MB，最好不要调用 string()方法以避免占用过多内存，这种情况下可以考虑将数据当作 Stream 流处理。具体的代码如下：

```
private final OkHttpClient client = new OkHttpClient();

public void get(String url) {
    Request request = new Request.Builder()
            .url(url)
            .build();

    Response response = null;
    try {
        response = client.newCall(request).execute();
        if (!response.isSuccessful()) throw new IOException("Unexpected code " + response);

        ResponseBody responseBody = response.body();
        if (responseBody.contentLength() > 1024 * 1024) {
```

```
                //内容大于 1MB 时，转化为流，然后进行相关处理
                InputStream inputStream = responseBody.byteStream();
            } else {
                //内容小于等于 1MB 时，使用 toString()方法将之转化为字符串，然后进行处理
                responseBody.toString();
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
```

2. 异步 GET 请求

使用异步 GET 请求的步骤与同步 GET 请求在创建 Call 对象之前都是相同的，之后的步骤如下：需要执行 Call 对象的 enqueue 方法，该方法接收一个 okhttp3.Callback 对象，enqueue 方法不会阻塞当前线程，会新开一个工作线程，让实际的网络请求在工作线程中执行；当异步请求成功后，会回调 Callback 对象的 onResponse 方法，在该方法中可以获取 Response 对象。当异步请求失败或者调用了 Call 对象的 cancel 方法时，会回调 Callback 对象的 onFailure 方法。onResponse()和 onFailure()这两个方法都是在工作线程中执行的。具体的代码如下：

```
    public void asyncGet(String url) {
        Request request = new Request.Builder()
                .url(url)
                .build();

        client.newCall(request).enqueue(new Callback() {
            @Override
            public void onFailure(Call call, IOException e) {
                e.printStackTrace();
            }

            @Override
            public void onResponse(Call call, Response response) throws IOException {
                if (!response.isSuccessful()) throw new IOException("Unexpected code " + response);
                ResponseBody responseBody = response.body();
                if (responseBody.contentLength() > 1024 * 1024) {
                    //内容大于 1MB 时，转化为流，然后进行相关处理
                    InputStream inputStream = responseBody.byteStream();
                } else {
                    //内容小于等于 1MB 时，使用 toString()方法将之转化为字符串，然后进行处理
                    responseBody.toString();
                }
            }
        });
    }
```

3. POST 发送键值对（Form 表单信息）

如果想用 POST 发送键值对字符串，可以在 post()方法中传入 FormBody 对象。FormBody 继承自 RequestBody，类似于 Web 前端中的 Form 表单。可以通过 FormBody.Builder 构建 FormBody。其他部分代码与 GET 请求相似，代码如下：

```
public void postForm(String url, Map<String, String> map) {
    FormBody.Builder builder = new FormBody.Builder();
    //将参数加入 FormBody 中，这样将成为请求报文的 body 部分
    for (Map.Entry<String, String> entry : map.entrySet()) {
        builder.add(entry.getKey(), entry.getValue());
    }
    RequestBody formBody = builder.build();
    Request request = new Request.Builder()
            .url(url)
            .post(formBody)
            .build();

    Response response = null;
    try {
        response = client.newCall(request).execute();
        if (!response.isSuccessful()) throw new IOException("Unexpected code " + response);
        //post 请求的响应一般较小，可以直接使用 String 类型处理
        response.body().string();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

代码内容容易理解，不再多说。这里有一点值得注意，在发送数据之前，Android 会对非 ASCII 码字符调用 encodeURIComponent 方法进行编码，如果传递的参数中有空格，空格符就会被编码成%20，服务器端会将其自动解码。

4. POST 上传文件

在 Android 中通过网络传输文件是经常发生的，使用 OkHttp 可以很容易处理文件上传的问题。但是这里会涉及 OkHttp 中的一个概念——MediaType。所以在讲解如何上传文件之前先对 MediaType 进行解释。

MediaType 指的是要传递数据的 MIME 类型，MediaType 对象包含 3 种信息：type、subtype 以及 CharSet，一般将这些信息传入 parse()方法中，这样就能解析出 MediaType 对象，比如“text/x-markdown; charset=utf-8”，type 值是 text，表示是文本这一大类；/后面的 x-markdown 是 subtype，表示是文本这一大类下的 markdown 小类；charset=utf-8 则表示采用 UTF-8 编码。如果不知道某种类型数据的 MIME 类型，可以参见连接 Media Types 和 MIME 参考手册（较

详细地列出了所有数据的 MIME 类型）。以下是几种常见数据的 MIME 类型值：json 类型的 application/json；xml 类型的 application/xml；png 类型的 image/png；jpg 类型的 image/jpeg；gif 类型的 image/gif。在我们上传文件时，需要根据文件类型判断合适的 MediaType。

下面通过一个上传 jpg 图片的方法来展示如何上传文件，代码如下

```
public static final MediaType MEDIA_TYPE_MARKDOWN
        = MediaType.parse("image/jpeg");
public void postFile(String url, String path) {
    File file = new File(path);
    Request request = new Request.Builder()
            .url(url)
            .post(RequestBody.create(MEDIA_TYPE_MARKDOWN, file))
            .build();

    Response response = null;
    try {
        response = client.newCall(request).execute();
        if (!response.isSuccessful()) throw new IOException("Unexpected code " + response);
        //POST 请求的响应一般较小，可以直接使用 String 类型处理
        response.body().string();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

如果需要上传的是其他类型，只需要将类型替换掉即可。当然，如果是文本类型就需要像“text/x-markdown; charset=utf-8”这种格式，在“text/x-markdown”后面加入编码格式“charset=utf-8”。

5. POST 发送多样式数据

在有些应用中，会有类似于 Web 中表单的界面，同时上传文件以及传递键值对参数。这种情况 OkHttp 也提供了解决方案。具体来说，就是使用 MultipartBody.Builder 的 setType()方法设置 MultipartBody 的 MediaType 类型，一般情况下，将该值设置为 MultipartBody.FORM，即 W3C 定义的 multipart/form-data 类型。然后通过 MultipartBody.Builder 的方法 addFormDataPart(String name, String value)或 addFormDataPart(String name, String filename, RequestBody body)添加数据，其中前者添加的是字符串键值对数据，后者可以添加文件。至于接收响应信息的处理与其他请求一样，代码如下：

```
private static final String IMGUR_CLIENT_ID = "...";
private static final MediaType MEDIA_TYPE_PNG = MediaType.parse("image/png");

public void postMul(String url, String path, Map<String, String> map) {
    MultipartBody.Builder builder = new MultipartBody.Builder();
```

```
        //W3C 定义的 multipart/form-data 类型
        builder.setType(MultipartBody.FORM);
        for (Map.Entry<String, String> entry : map.entrySet()) {
            builder.addFormDataPart(entry.getKey(), entry.getValue());
        }
        builder.addFormDataPart("image", "logo-square.png",
                RequestBody.create(MEDIA_TYPE_PNG, new File(path)));
        RequestBody requestBody = builder.build();

        Request request = new Request.Builder()
                .header("Authorization", "Client-ID " + IMGUR_CLIENT_ID)
                .url(url)
                .post(requestBody)
                .build();

        Response response = null;
        try {
            response = client.newCall(request).execute();
            if (!response.isSuccessful()) throw new IOException("Unexpected code " + response);
            System.out.println(response.body().string());
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
```

6. 取消请求

当请求不再需要的时候，我们应该中止请求，比如退出当前的 Activity，那么在 Activity 中发出的请求将被中止。可以通过调用 Call 的 cancel()方法立即中止请求，如果线程正在写入 Request 或读取 Response，就会抛出 IOException 异常。同步请求和异步请求都可以被取消。取消请求的方法很简单，只需要 call.cancel()即可，但是如何去把握取消请求的时机很重要，对此读者应多加关注。

7. 缓存问题

在 Android 开发中缓存问题是非常重要的。采用缓存，既可以大大缓解数据交互的压力、提高客户端的响应速度、减少服务器压力，又能提供一定的离线浏览。在 OkHttp 中为了缓存响应，需要创建可以读写的缓存目录和缓存大小的限制。这个缓存目录应该是私有的，不信任的程序应不能读取缓存内容。一个缓存目录同时拥有多个缓存访问是错误的。大多数程序只需要调用一次 new OkHttp()，在第一次调用时配置好缓存，然后只需在要其他地方调用这个实例就可以了，否则两个缓存示例会互相干扰，破坏响应缓存，而且有可能会导致程序崩溃。响应缓存使用 HTTP 头作为配置，可以在请求头中添加“Cache-Control: max-stale=3600”，OkHttp 缓存会支持。服务将通过响应头确定响应缓存多长时间，例如使用 Cache-Control:

max-age=3600。如果想要强制使用网络或者缓存，可以在构建 Request 时用 Builder 对象调用 cacheControl()方法，并用 CacheControl.FORCE_NETWORK 或者 CacheControl.FORCE_CACHE 作为参数。缓存使用范例如下：

```
public class OkHttpUtil {

    private final OkHttpClient client;

    public OkHttpUtil(File cacheDirectory) {
        int cacheSize = 10 * 1024 * 1024;
        //设置大小以及路径
        Cache cache = new Cache(cacheDirectory, cacheSize);
        client = new OkHttpClient.Builder()
                .cache(cache)
                .build();
    }
......
}
```

上述代码通过构造函数来设置缓存，如果想要使用缓存，可以使用 response.cacheResponse()。

10.3.3 OkHttp 使用实例

通过上一部分的介绍，读者应该已经能够基本使用 OkHttp 了。下面我们通过一个登录、注册实例来使用上面的写方法。

首先创建一个项目 OkHttp，修改 MainActivity 的布局文件 activity_main.xml：

```
package com.buaa.okhttp;

import android.app.Activity;
import android.content.Intent;
import android.os.Handler;
import android.os.Message;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;

import java.util.HashMap;
import java.util.Map;

public class MainActivity extends AppCompatActivity implements View.OnClickListener {
```

```
    private Button login;
    private Button register;
    private EditText name;
    private EditText psd;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        initView();
    }

    private void initView() {
        login = (Button) findViewById(R.id.login_button);
        register = (Button) findViewById(R.id.register_button);
        login.setOnClickListener(this);
        register.setOnClickListener(this);

        name = (EditText) findViewById(R.id.login_name);
        psd = (EditText) findViewById(R.id.login_password);
    }

    @Override
    public void onClick(View v) {
        switch (v.getId()) {
            case R.id.login_button:
                new Thread(new LoginByOkHttp()).start();
                break;
            case R.id.register_button:
                //这里只是为了页面布局优美，不做业务上的处理
                Toast.makeText(MainActivity.this, "敬请期待", Toast.LENGTH_LONG).show();
                break;
        }
    }

    Handler handler = new Handler() {
        @Override
        public void handleMessage(Message msg) {
            super.handleMessage(msg);
            switch (msg.what) {
                case 1:
                    Toast.makeText(MainActivity.this, "登录成功", Toast.LENGTH_LONG).show();
```

```
                        BookActivity.startActivity(MainActivity.this);
                        break;
                    case 2:
                        Toast.makeText(MainActivity.this, "密码不正确，请重新输入",
                        Toast.LENGTH_LONG).show();
                        break;
                    case 3:
                        Toast.makeText(MainActivity.this, "用户名不存在，您可以先注册",
                        Toast.LENGTH_LONG).show();
                        break;
                }
            }
        };

        private class LoginByOkHttp implements Runnable {
            @Override
            public void run() {
                //使用 OkHttpUtil 类，初始化 OkHttpClient 以及缓存
                OkHttpUtil okHttpUtil = new OkHttpUtil(getCacheDir() + "/ok");
                Map<String, String> map = new HashMap<>();
                map.put("name", name.getText().toString());
                map.put("psd", psd.getText().toString());
                String result = okHttpUtil.postForm("http://192.168.1.104:8080", map);
                if (result.equals("1")) {
                    handler.sendEmptyMessage(1);
                } else if (result.equals("2")) {
                    handler.sendEmptyMessage(2);
                } else {
                    handler.sendEmptyMessage(3);
                }
            }
        }
    }
```

MainActivity 中调用的 postForm()方法只是将上一部分的方法改为有返回值的方法，其他不变，这里不再展示。接下来创建 BookActivity，并当成功登录后进入 BookActivity，可以在此 Activity 中阅读文章。下面先修改布局文件：

```
    <?xml version="1.0" encoding="utf-8"?>
    <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:tools="http://schemas.android.com/tools"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical"
```

```
        tools:context="com.buaa.okhttp.BookActivity">

        <TextView
            android:id="@+id/title"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:gravity="center"
            android:textSize="26sp" />

        <ScrollView
            android:layout_width="match_parent"
            android:layout_height="wrap_content">

            <TextView
                android:id="@+id/book"
                android:layout_width="match_parent"
                android:layout_height="wrap_content" />
        </ScrollView>

    </LinearLayout>
```

然后在 BookActivity 中获取 TextView 控件，并将通过网络请求获取的值放置到 TextView 中，代码如下：

```
package com.buaa.okhttp;

import android.app.Activity;
import android.content.Intent;
import android.os.Handler;
import android.os.Message;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.TextView;

import org.json.JSONException;
import org.json.JSONObject;

public class BookActivity extends AppCompatActivity {
    private String result;
    private TextView context;
    private TextView title;
    Handler handler = new Handler() {
        @Override
        public void handleMessage(Message msg) {
```

```
                super.handleMessage(msg);
                if (msg.what == 1) {
                    try {
                        JSONObject jsonObject = new JSONObject(result);
                        title.setText(jsonObject.getString("name"));
                        context.setText(jsonObject.getString("context"));
                    } catch (JSONException e) {
                        e.printStackTrace();
                    }
                }
            }
        };

        @Override
        protected void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            setContentView(R.layout.activity_book);
            new Thread(new Runnable() {
                @Override
                public void run() {
                    OkHttpUtil okHttpUtil = new OkHttpUtil(getCacheDir() + "/ok");
                    result = okHttpUtil.get("http://192.168.1.104:8080");
                    handler.sendEmptyMessage(1);
                }
            }).start();
            context = (TextView) findViewById(R.id.book);
            title = (TextView) findViewById(R.id.title);
        }

        //开启 Activity 的方法，可以在其他 Activity 中调用
        public static void startActivity(Activity activity) {
            Intent intent = new Intent("android.intent.action.Register");
            activity.startActivity(intent);
        }
    }
```

网络通信使用的是前一部分的 get()方法，只是将其修改为有返回值的方法而已，此处不再赘述。由于服务器端返回的值是 JSON 格式的数据（做了一点修改，与 10.2 节的返回值不同），因此这里使用 JsonObject 对其进行解析，获取了键值为“name”和“context”的值，并分别将它们放置到 TextView 中。

至此，实例代码完成，但是还要在 AndroidManifest.xml 中申请权限：

```
<uses-permission android:name="android.permission.INTERNET" />
```

运行程序，当在登录界面输入正确的用户密码时，就会跳转到文章展示界面。关于 OkHttp 的内容还有很多，不是一节内容就可以讲解完的，只有在使用中不停地摸索，才能快速掌握。学习编程的最好方法是进行编程，而不是靠书本，所以希望读者能够将各种通信方式都练习一遍。

10.3.4 JSON 简介

在 10.3.2 小节使用到了 JSON 格式的数据，部分读者可能对 JSON 并不熟悉。JSON（JavaScript Object Notation）是一种轻量级的数据交换格式，具有良好的可读和便于快速编写的特性。业内主流技术为其提供了完整的解决方案（有点类似于正则表达式，获得了当今大部分语言的支持），从而可以在不同平台间进行数据交换。简单地说，它就是一种有着特定格式的文本。

JSON 具备对象和数组这两种结构，通过这两种结构相互组合可以表示各种复杂的结构。

（1）对象：对象是通过“{}”括起来的内容，数据结构为 {key: value,key: value,...}的键值对。在面向对象的语言中，key 为对象的属性，value 为对应的属性值，所以很容易理解，这个属性值的类型可以是数字、字符串、数组、对象等。

（2）数组：数组在 JSON 中是中括号“[]”括起来的内容，数据结构为 ["java","javascript","vb",...]，取值方式和所有语言中一样，使用索引获取，字段值的类型可以是数字、字符串、数组、对象等。

在 10.3.3 小节我们接收到的 JSON 是一个 JSON 对象，通过对它的解析，我们可以还原出原始模样：

```
{"name":"书的名字","context":"书的内容"}
```

下面通过几个 JSON 对象和 JSON 数组的示例来加深对它们的印象与理解。

（1）简单对象：{"firstName":"Brett","lastName":"McLaughlin","email":"aaaa"}

（2）简单数组：["java","javascript","vb"]

（3）包含对象的数组：[{"name":"张三","age":18},"北京"，"南京"]

（4）包含数组的对象：{name:"安徽省",cities:[{name:"芜湖市",quxian:["繁昌县","芜湖县","南陵县","三山区"]},{name:"合肥市",quxian:["肥西县","蜀山区","庐阳区"]}]}

在 Android 中对 JSON 数组或者 JSON 对象的解析主要依靠 org.json 包下的 JSONObject、JSONArray 这两个类，一个代表 JSON 对象，一个代表 JSON 数组，通过它们可以很容易地构建与解析 JSON 格式的数据。下面通过两段代码来说明在 Android 中 JSON 的构建与解析。构建 JSON 对象数组的范例代码如下：

```
public void createJSON() {
    try {
        //实例化一个 JSON 对象
        JSONObject jsonObject1 = new JSONObject();
        jsonObject1.put("name","李子熟了");
```

```
            jsonObject1.put("address","上海");

            JSONObject jsonObject2 = new JSONObject();
            jsonObject2.put("name","李子熟了");
            jsonObject2.put("address","上海");

            //创建一个 JSON 数组
            JSONArray jsonArray = new JSONArray();
            //将 JSON 对象加入数组中
            jsonArray.put(0,jsonObject1);
            jsonArray.put(1,jsonObject2);
        } catch (JSONException e) {
            e.printStackTrace();
        }
    }
```

解析 JSON 格式数据的范例：

```
public void handlerJSON(String json) {
    Map<String, Object> map = new HashMap<>();
    try {
        /**
          * 将接收到的文本作为参数传入构造函数，这样就产生了一个 JSON 对象
          * JSON 数组的也是同样道理
          */
        JSONObject jsonObject = new JSONObject(json);
        String bookName = jsonObject.getString("name");

        map.put("name", bookName);
        List<String> list = new ArrayList<>();
        //获取 JSON 对象中的 JSON 数组
        JSONArray jsonArray = jsonObject.getJSONArray("city");
        //就像使用数组一样
        for (int i = 0; i < jsonArray.length(); i++) {
            list.add(jsonArray.getString(i));
        }
        map.put("city",list);
    } catch (JSONException e) {
        e.printStackTrace();
    }
}
```

关于 JSON 的内容就讲到这里。下一节我们将讲解如何使用 Socket 进行网络通信。

10.4　使用 Socket 进行网络通信

10.4.1　Socket 简介

Socket（套接字）是对 TCP/IP 协议的封装和应用，根据底层封装协议的不同，Socket 的类型可以分为流套接字（streamsocket）和数据报套接字（datagramsocket）两种。流套接字将 TCP 作为端对端协议，提供了一个可信赖的字节流服务；数据报套接字使用 UDP 协议，提供数据打包发送服务，应用程序可以通过它发送最长 64KB 的信息。Socket 的通信模型图如图 10-5 所示。

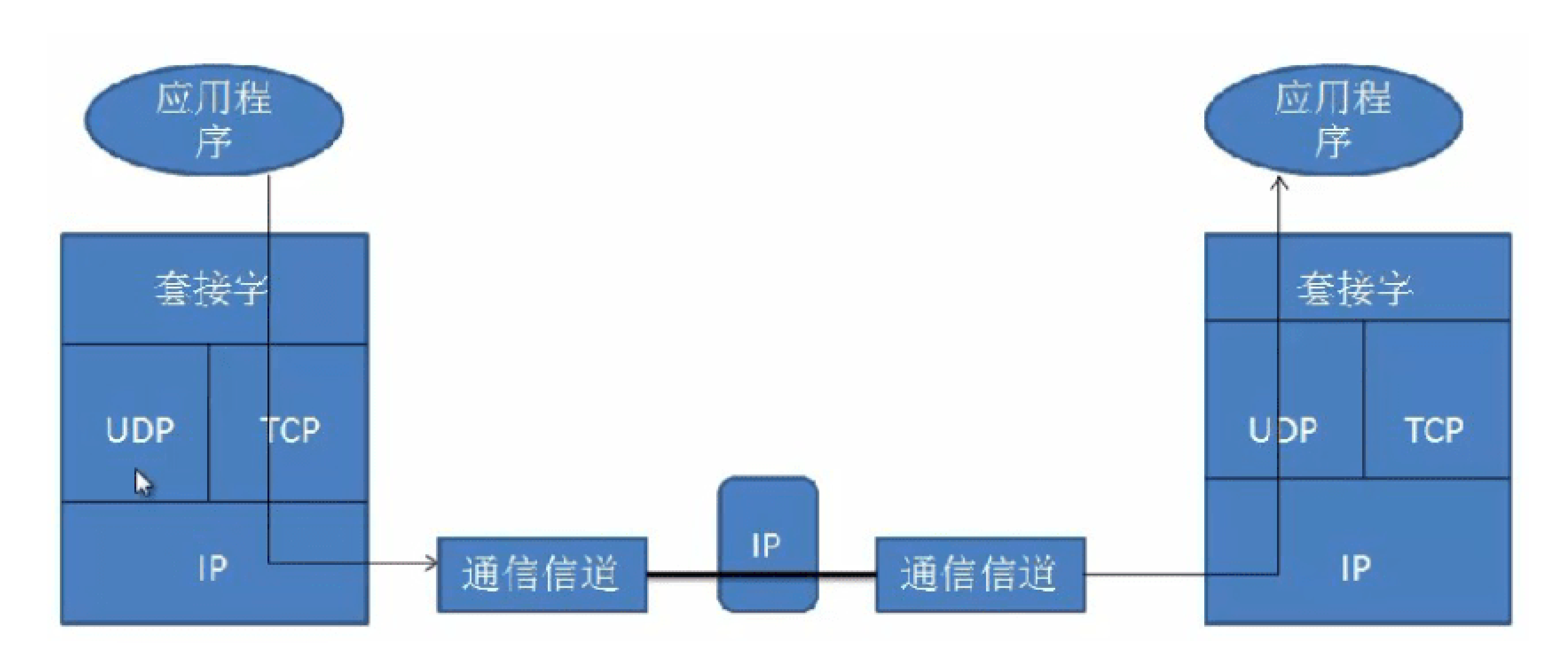


图 10-5　Socket 的通信模型图

通过图 10-5 可以很容易地看出，使用 Socket 进行两个应用程序之间的通信时可以选择使用 TCP 还是 UDP 作为其底层协议。对比两种方式，就会发现它们各有优劣，TCP 首先连接接收方，然后发送数据，保证成功率，速度相对较慢（相比 HTTP 方式还是非常快的）；UDP 把数据打包成数据包，然后直接发送对应的 IP 地址，速度快，但是不保证成功率，并且数据大小有限。

一个功能齐全的 Socket，都要包含以下基本结构，其工作过程包含 4 个基本的步骤：创建 Socket，打开连接到 Socket 的输入/出流，按照一定的协议对 Socket 进行读/写操作，关闭 Socket。

Java 在 java.net 包中提供了 Socket 和 ServerSocket 两个类，分别用来表示双向连接的客户端和服务端，是 Socket 编程的核心类。构造方法很多，一般情况下使用下面两种：

```
Socket client = new Socket("127.0.0.1    ", 9999);
ServerSocket server = new ServerSocket(9999);
```

其中，Socket 类用于实例化一个 Client，参数分别是要访问的 IP 地址和端口号，这个端口号要与服务端一致。ServerSocket 类用于实例化一个 Server，其中的参数用来设置端口号。这里的端口不能与“3306”“80”“8080”等常用端口号冲突。

下面以基于 TCP 的 Socket 为例来讲解如何使用 Socket。

10.4.2　基于 TCP 的 Socket

使用 Java Socket 创建一个服务端程序，并运行在 PC 上，然后在手机上编写客户端程序，在局域网内访问服务端。下面先编写服务端，代码如下：

```
package com.buaa;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.OutputStream;
import java.net.ServerSocket;
import java.net.Socket;

public class Server{
    public static void main(String[] args) throws IOException {
        ServerSocket serivce = new ServerSocket(30000);
        while (true) {
            // 等待客户端连接
            Socket socket = serivce.accept();
            new Thread(new AndroidRunable(socket)).start();
        }
    }
}

class AndroidRunable implements Runnable {

    Socket socket = null;

    public AndroidRunable(Socket socket) {
        this.socket = socket;
    }

    public void run() {
        // 向 Android 客户端输出 hello world
        String line = null;
        InputStream input;
        OutputStream output;
        String str = "hello world!";
        try {
            //向客户端发送信息
            output = socket.getOutputStream();
```

```
                input = socket.getInputStream();
                BufferedReader bff = new BufferedReader(
                        new InputStreamReader(input));
                output.write(str.getBytes("gbk"));
                output.flush();
                //半关闭 Socket
                socket.shutdownOutput();
                //获取客户端的信息
                while ((line = bff.readLine()) != null) {
                    System.out.print(line);
                }
                //关闭输入输出流
                output.close();
                bff.close();
                input.close();
                socket.close();

            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
```

这里的代码很简单，单纯地使用 ServerSocket 建立服务，设置端口号为 30000，然后每当有客户端访问时就返回一个“hello world”。编辑完成服务端之后，我们在 Android Studio 中创建一个用于创建 Socket 客户端的类，代码如下：

```
public class SocketUtil {

    private String str;
    private Socket socket;
    private String ip;

    public SocketUtil(String str, String ip) {
        this.str = str;
        this.ip = ip;
    }

    public String sendMessage() {
        String result = "";
        try {
            socket = new Socket();
            socket.connect(new InetSocketAddress(ip, 30000), 5000);
```

```
                OutputStream out = socket.getOutputStream();
                out.write(str.getBytes());
                out.flush();

                BufferedReader bff = new BufferedReader(new InputStreamReader(
                        socket.getInputStream()));
                String line = null;
                StringBuffer buffer = new StringBuffer();
                while ((line = bff.readLine()) != null) {
                    buffer.append(line);
                }
                result = buffer.toString();
                bff.close();
                out.close();
                socket.close();
            } catch (SocketTimeoutException e) {
                //连接超时，在 UI 界面显示消息
                Log.i("socket", e.toString());
            } catch (IOException e) {
                e.printStackTrace();
            }
            return result;
        }
    }
```

在本类中，使用 Socket 连接服务端，然后发送相关信息并接收服务端数据。代码并不难，下面就在 MainActivity 中使用此类。当然，使用它之前要先修改 MainActivity 的布局文件 activity_main.xml，代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context="com.buaa.socket.MainActivity">

    <EditText
        android:id="@+id/message"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:paddingTop="40dp"
        android:textSize="24sp" />
```

```
    <Button
        android:id="@+id/button"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="20dp"
        android:text="发送消息" />

    <TextView
        android:id="@+id/book"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="20dp" />
</LinearLayout>
```

这里使用一个 EditText 来编辑向服务端发送的内容，并用一个 Button 来点击触发网络通信的事件，然后用一个 TextView 展示服务端返回的值。接下来在 MainActivity 中通过 findViewById()方法来获取这些控件，并设置 Button 的点击事件，代码如下：

```
public class MainActivity extends AppCompatActivity {

    private Button button;
    private TextView textView;
    private String result;
    private Handler handler = new Handler() {
        @Override
        public void handleMessage(Message msg) {
            super.handleMessage(msg);
            if (msg.what == 123) {
                textView.setText(result);
            }
        }
    };

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        initView();
    }

    private void initView() {
        textView = (TextView) findViewById(R.id.book);
        final EditText editText = (EditText) findViewById(R.id.message);
        button = (Button) this.findViewById(R.id.button);
```

```
                button.setOnClickListener(new View.OnClickListener() {
                    @Override
                    public void onClick(View v) {
                        new Thread(new Runnable() {
                            @Override
                            public void run() {
                                result = new SocketUtil(editText.getText().toString(),
"192.168.1.104").sendMessage();
                                handler.sendEmptyMessage(123);
                            }
                        }).start();
                    }
                });
            }
        }
```

在 Activity 中只是对点击事件做了处理，并将服务端返回的值展示在 TextView 上。添加完网络权限之后，运行程序，在 EditText 中输入内容，然后点击“发送消息”按钮，将“你好”发送到服务端，并接收到服务端返回的“Hello World”，如图 10-6 所示。

观察服务端代码所在的控制台，发现也确实接收到了手机发送的内容，如图 10-7 所示。



图 10-6　Socket 的通信：客户端发送并接收响应的消息

Problems　Javadoc　Declaration　Console
Server [Java Application] D:\java_1.6\jre\bin\javaw.exe (2016-
你好

图 10-7　Socket 通信：服务端接收到消息

到此读者可能会发现，相比于直接使用 HTTP 进行编程，Socket 确实相对麻烦一些。但是这种麻烦带来的是一些效率的提升，并可以应对一些特殊的需求，比如说 P2P。当然也有读者会提出疑惑，之前的几节我们都是使用 Tomcat 作为服务器的容器，这样编程就会很简单，本节为什么不使用呢？答案很简单，Tomcat 的实现也是基于 HTTP 的，所以如果 Socket 想要与 10.3 节的服务端进行通信，就必须在通信时构建符合 HTTP 规范的请求头，代码如下：

```
socket = new Socket(host, 8080);
OutputStream os = socket.getOutputStream();
```

```
StringBuffer head = new StringBuffer();
head.append("GET / HTTP/1.1    " + SEQUENCE);
head.append("Host:" + host + SEQUENCE + SEQUENCE);
head.append("Accept:text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8");
head.append("Accept-Language:zh-CN,zh;q=0.8");
head.append("User-Agent:Mozilla/5.0 (Windows NT 6.2; WOW64) AppleWebKit/537.11 (KHTML, like
Gecko) Chrome/23.0.1271.95 Safari/537.11");
os.write(head.toString().getBytes());
os.flush();
InputStream is = socket.getInputStream();
```

这样其实就相当于实现了 HTTP，此时直接使用 HTTP 进行通信会更好。另外，本例是基于 TCP 来实现 Socket 的，而在一些即时通信应用中，使用 UDP 更加广泛。读者如果有兴趣，可以自己动手实现基于 UDP 的 Socket。

10.5　WebView

除了 HTTP 通信与 Socket 通信两种主要的网络技术外，在 Android 中还提供了一种加载和显示网页的技术——WebView。这可以让我们去处理一些特殊的需求，比如像微信那样在应用程序里展示网页，或者说使用 WebView 来为 UI 界面布局。

10.5.1　WebView 的基本使用

WebView 的使用非常简单，新建一个项目 internet，修改 activity_main.xml 中的代码，加入一个 WebView 控件。WebView 控件是一个新的控件，用于显示网页，为了可以在 Activity 中获取 WebView 而设置了 id，代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.buaa.internet.MainActivity">

    <WebView
        android:id="@+id/web"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />
</RelativeLayout>
```

Activity 的内容很简单，只是通过 findViewById()方法获取了 WebView 实例，并使用 webView.loadUrl("http://www.baidu.com")将链接“http://www.baidu.com”（百度首页）加载到了布局中，代码如下：

```
package com.buaa.internet;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.webkit.WebSettings;
import android.webkit.WebView;
import android.webkit.WebViewClient;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        WebView webView = (WebView) findViewById(R.id.web);
        //获取 WebSettings 类的实例，此类用于对 WebView 加载的网页进行设置
        WebSettings webSettings = webView.getSettings();
        //使 WebView 可以使用 JavaScript
        webSettings.setJavaScriptEnabled(true);
        //请求加载百度，并交由 WebClient 去处理
        webView.loadUrl("http://www.baidu.com");
        //使用 WebViewClient 设置监听并处理 WebView 的请求事件
        webView.setWebViewClient(new WebViewClient() {
            @Override
            public boolean shouldOverrideUrlLoading(WebView view, String
                    url) {
                //根据 url 真正去加载网页的操作
                view.loadUrl(url);
                // 在当前 WebView 中打开网页，而不在浏览器中
                return true;
            }
        });
    }
}
```

Activity 中的操作并不多，正如注释中描述的，WebView 的 getSettings()方法获取 WebSettings 的实例，然后去设置一些属性。此处只是调用了 setJavaScriptEnabled()方法来让 WebView 可以使用 JavaScript 脚本。然后使用 webView.loadUrl("http://www.baidu.com")来加载网页，但是此时并不是真正执行网页的加载动作，只是发送了一个请求，真正的动作是通过 WebClient 来完成的。最后实现 WebView 的 setWebViewClient()方法，在匿名内部类中处理真正的加载操作。

完成这些操作之后，在 AndroidManife.xml 中加入网络权限：

```
<uses-permission android:name="android.permission.INTERNET"/>
```

运行程序，就可以打开一个百度页面了，如图 10-8 所示。

和之前一样，这里需要注意保证手机有网络连接，使用模拟器时需要保证计算机有网络连接。



图 10-8 使用 WebView 打开网页

10.5.2 使用 HTML 进行 UI 设计

一般 Android 的 UI 使用的是 XML。使用 XML 制作很高级的 UI 会很复杂，如果使用 HTML 来进行 UI 设计就会简单很多。

具体来说，只需要开发一个符合 UI 要求的 HTML 文档并放入 assets 文件中，然后加载此 HTML 文件即可。使用 Android Studio 进行开发时，需要自己首先创建出 assets 文件夹。创建方法很简单，右击“main”，选择“new”，然后选择“folder”，选择新建“aseets folder”即可。然后在 assets 中新建一个 HTML 文档，或者将已有的 HTML 文档放入其中，再通过 webView.loadUrl("file:///android_asset/user.html")将我们自己创建的 HTML 文档加载进页面。为让读者能够有更直观的感受，下面通过一个实例来说明它的相关用法。

直接在本节上一实例的基础上进行修改，activity_main.xml 布局文件不改，MainActivity 类修改如下：

```
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        WebView webView = (WebView) findViewById(R.id.web);
        //获取 WebSettings 类的实例，此类用于对 WebView 加载的网页进行设置
        WebSettings webSettings = webView.getSettings();
        //使 WebView 可以使用 JavaScript
        webSettings.setJavaScriptEnabled(true);
        //请求加载 html 页面，并交由 WebClient 去处理
        webView.loadUrl("file:///android_asset/user.html");
        webView.addJavascriptInterface(new JavaScriptInterface(), "AndroidWebView");

        //使用 WebViewClient 设置监听并处理 WebView 的请求事件
        webView.setWebViewClient(new WebViewClient() {
            @Override
            public boolean shouldOverrideUrlLoading(WebView view, String
                    url) {
                //根据 url 真正去加载网页的操作
```

```
                view.loadUrl(url);
                //表示当前 WebView 可以处理打开新网页的请求
                return true;
            }
        });
    }

    private class JavaScriptInterface {
        @JavascriptInterface
        public String showUser() {
            JSONObject map;
            JSONArray array = new JSONArray();
            try {
                map = new JSONObject();
                map.put("0", "李宗盛");
                map.put("1", 45);
                map.put("2", "中国");
                array.put(map);

                map = new JSONObject();
                map.put("0", "梁静茹");
                map.put("1", 32);
                map.put("2", "马来西亚");
                array.put(map);

                map = new JSONObject();
                map.put("0", "奥巴马");
                map.put("1", 60);
                map.put("2", "美国");
                array.put(map);

                map = new JSONObject();
                map.put("0", "李子熟了");
                map.put("1", 60);
                map.put("2", "中国");
                array.put(map);
            } catch (JSONException e) {
                e.printStackTrace();
            }
            return array.toString();
        }

        @JavascriptInterface
```

```
            public void submit(String context) {
                Toast.makeText(MainActivity.this, context, Toast.LENGTH_LONG).show();
            }
        }
    }
```

与之前相比，增加了一个内部类 JavaScriptInterface，其中有两个方法，而在这两个方法上面都加入了注解“@JavascriptInterface”，这表明它们可以在 JavaScript 中使用。细心的读者会发现在 onCreate()方法中加入了一行代码：

```
webView.addJavascriptInterface(new JavaScriptInterface(), "AndroidWebView");
```

这行代码的作用就是为当前的 WebView 添加一个 JavaScript 接口，使 WebView 可以使用 JavaScriptInterface 类中的两个方法。使用时格式如下：

```
window.AndroidWebView.showUser()
```

了解了这些之后，就可以编写一个 HTML 文档了。创建 user.html：

```
<html>
    <head>
        <title></title>
        <style type="text/css">
            body,table{
            font-size:12px;
            }
            table{
            table-layout:fixed;
            empty-cells:show;
            border-collapse: collapse;
            margin:0 auto;
            }
            td{
            height:40px;
            }
            h1,h2,h3{
            font-size:12px;
            margin:0;
            padding:0;
            }
            .table{
            border:1px solid #cad9ea;
            color:#666;
            }
            .table th {
            background-repeat:repeat-x;
```

```
                height:40px;
                }
                .table td,.table th{
                border:1px solid #cad9ea;
                padding:0 1em 0;
                }
            </style>
            <script type="text/javascript">
                function showUser(){
                        var userList = window.AndroidWebView.showUser();
                        var obj = eval("("+userList+")");
                        var tab =document.getElementById('userTable');
                        for(var i=0;i<obj.length;i++){
                            var tr=document.createElement("tr");
                            for(var j=0;j<3;j++){
                                var td=document.createElement("td");
                                td.innerHTML=obj[i][j];
                                tr.appendChild(td);
                            }
                            tab.appendChild(tr);
                        }
                    }
                function submit(){
                    window.AndroidWebView.submit("这是 html 页面传递的数据");
                }

            </script>
        </head>
        <body onload="showUser()">
            <table width="96%" class="table" id="userTable">
                <tr>
                    <th>姓名</th>
                    <th>年龄</th>
                    <th>国籍</th>
                </tr>
            </table>
            <div align="center">
                <button onclick="submit()">提交</button>
            </div>
        </body>
    </html>
```

这里使用了一个表格，当页面加载完成时调用 JavaScriptInterface 类中的 showUser()方法，

并将数据以表格的形式展示出来，同时使用一个 button 标签，当点击它时，调用 submit(String context)方法。HTML 文档的讲解不是本书的重点，读者只需关注 window.AndroidWebView.showUser()和 window.AndroidWebView.submit("这是 html 页面传递的数据")这种调用 Java 方法的方式即可。

运行程序，想要展示的数据以一种比较清爽的方式展现在界面中，效果如图 10-9 所示。

当点击按钮时，就会弹出一个 Toast，效果如图 10-10 所示，表示在 Android 中可以使用 WebView 进行 HTML 页面、JavaScript、Java 之间的通信。

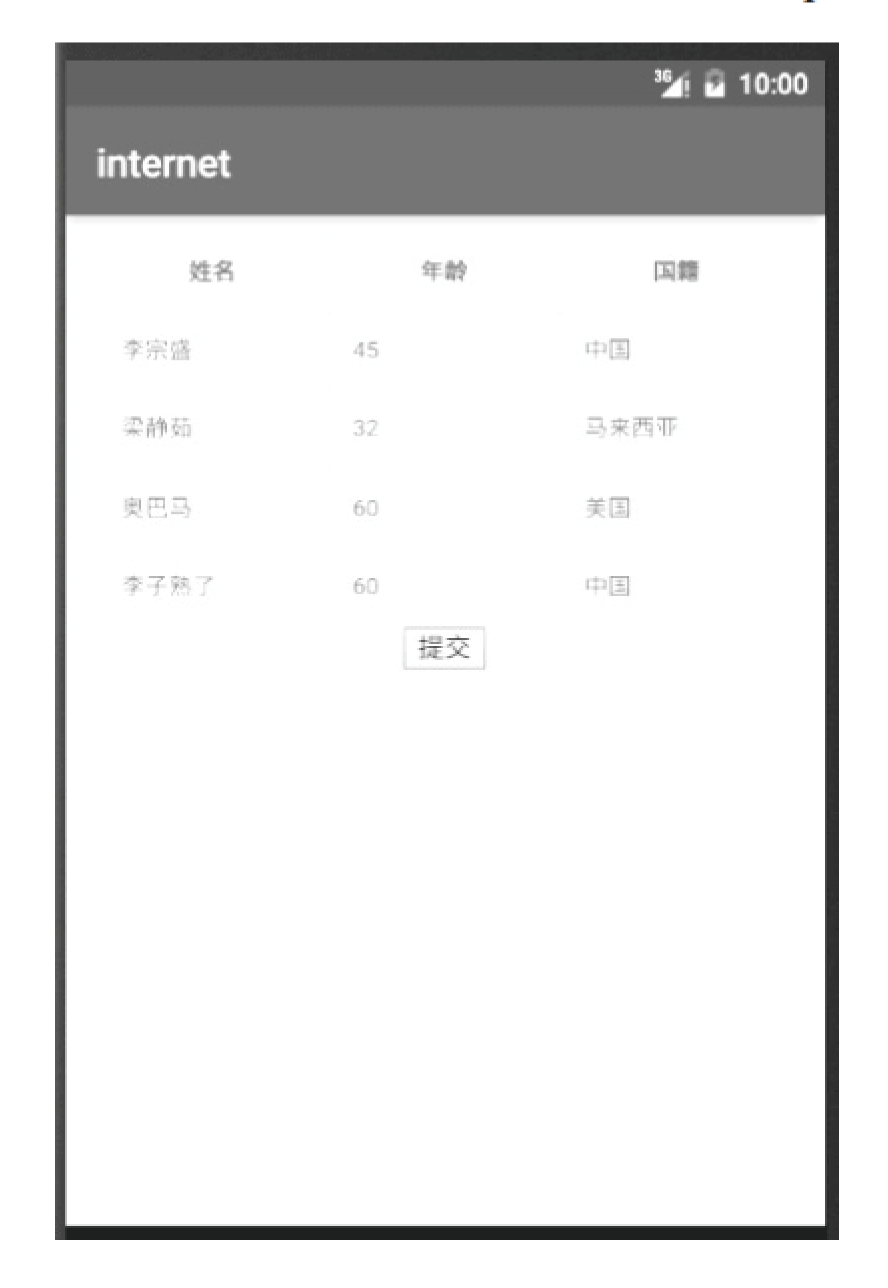


图 10-9　使用 HTML 代码进行布局

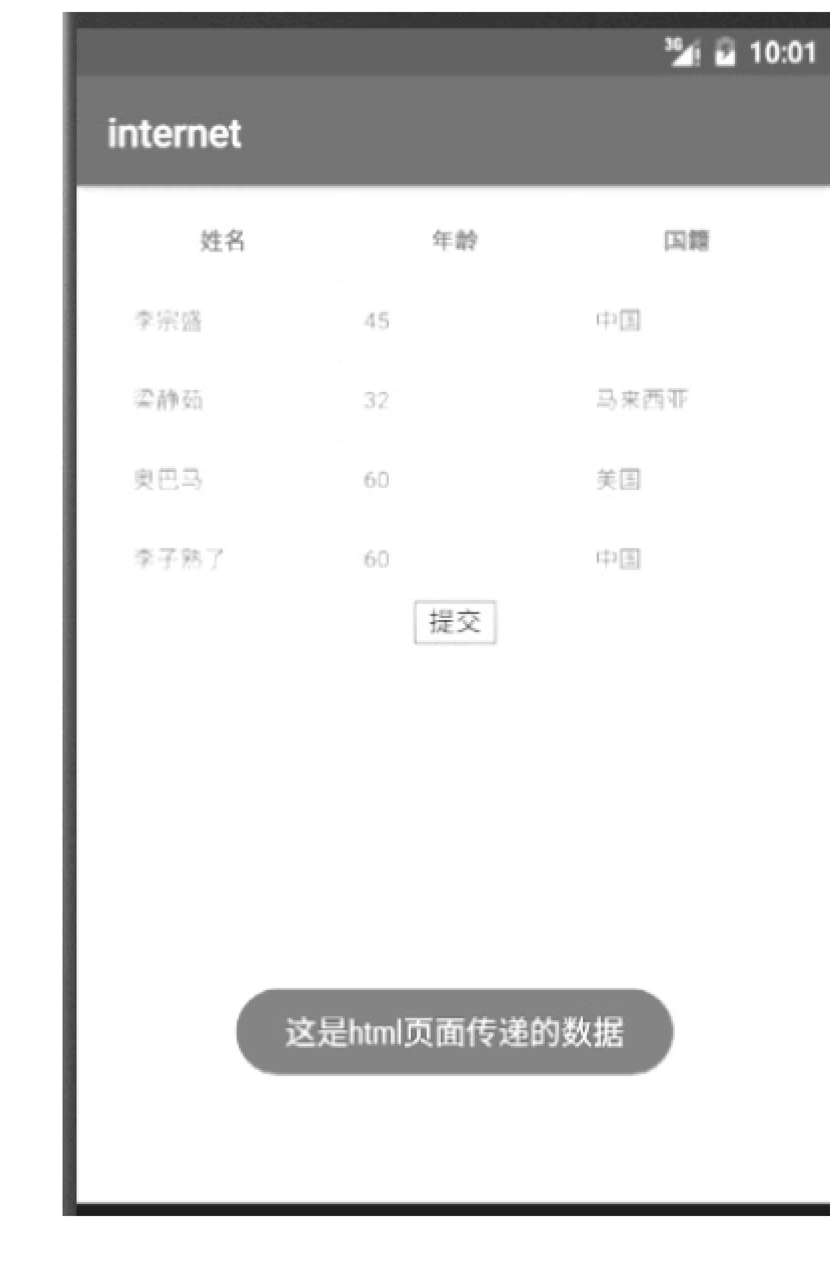


图 10-10　获取 HTML 传递的数据

显而易见，通过 HTML 文档的方式来进行布局更加简单容易，也能够容易地实现复杂的视觉要求。但是，通过运行程序发现速度比原生的 xml 布局要慢。因此，如果对性能有较高的要求，尽量不使用 HTML 文档的方式进行布局。而对一些使用较少、性能要求较低的界面，为了节省成本，可以使用这种方式布局快速开发。

10.6　小　　结

本章对 Android 中的网络通信技术进行了系统的分析与总结。讲解了如何使用 HTTP 以及 Socket 进行网络通信，同时针对一些特殊的需要讲解了 WebView 的使用。本章还重点介绍了 OkHttp 这一实际开发中经常使用、非常重要的 HTTP 请求框架。对一个应用来说，网络通信的重要性是无须赘言的，因此熟练掌握它是必需的，而熟练掌握一门技术最快的方法就是不断地联系。



第 11 章 多媒体开发

与过去的手机相比，智能手机提供了注入拍照、录视频、听音乐、看视频等众多的娱乐方式。这些娱乐方式少不了强大的多媒体功能支持，Android 系统在这一方面做得非常出色。它提供了一系列的 API，使得我们可以在程序中调用很多手机的多媒体资源，从而编写出更加丰富多彩的应用程序。本章我们将对 Android 中这些常用的多媒体功能使用技巧进行学习。除了这些可供娱乐的多媒体功能外，拨号、短信、通知、动画这些功能也属于广义上的多媒体功能，也是本章讲解的重点。

11.1 拨号功能与短信功能

拨打电话和收发短信是每个手机最基本的功能之一，即使是许多年前的老手机也都会具备这些功能。Android 是出色的智能手机操作系统，通过拨号管理器和短信管理器轻松实现这些功能。在开发过程中，也会遇到一些需要在应用中进行拨打电话和发送短信的场景，这时需要开发者使用 Android 提供的 API 来进行拨号和短信管理。本节就来介绍如何在应用中使用拨号功能和发送短信功能。

11.1.1 拨号的实现

在 Android 应用中，拨打电话的功能实现是非常简单的，这是因为在 Android 中系统自带了拨号应用。我们在应用中想要实现拨打电话功能，只需要调用系统的拨号功能即可。具体来说就是通过 startActivity(Intent intent)来打开拨号应用，其中的参数可以是一个 aciton 为“Intent.ACTION_DIAL”或者“Intent.ACTION_CALL”的 Intent。这两种不同的 action 分别对应不同的开启方式，前者的特点是，程序进入了拨号界面，但是实际的拨号是由用户点击实现的；后者的特点是直接进行拨号。下面通过一个实例来进行说明。创建一个新的项目 call，修改 MainActivity 和布局文件：

```
package com.buaa.call;

import android.content.Intent;
import android.net.Uri;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        initView();
    }

    private void initView() {
        final EditText number = (EditText) findViewById(R.id.phone_number);
        Button button = (Button) findViewById(R.id.call_but);
```

```
            button.setOnClickListener(new View.OnClickListener() {
                @Override
                public void onClick(View v) {
                    Intent callIntent = new Intent(Intent.ACTION_DIAL);
                    //Intent callIntent = new Intent(Intent.ACTION_CALL);
                    //这里向拨号器传递数据，在号码前面必须加上"tel:"
                    Uri data = Uri.parse("tel:" + number.getText().toString());
                    callIntent.setData(data);
                    startActivity(callIntent);
                }
            });
        }
    }
```

此时采用的 aciton 为“Intent.ACTION_DIAL”的方式来调用拨号应用。代码很简单，先获取 EditText 的输入内容，然后点击 Button 按钮时触发拨号事件。这里的 EditText 和 Button 是将布局文件修改后加入的控件，代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context="com.buaa.call.MainActivity">

    <EditText
        android:id="@+id/phone_number"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="请输入号码"
        android:inputType="phone"
        android:paddingTop="20dp"
        android:textSize="22sp" />

    <Button
        android:id="@+id/call_but"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="拨打电话" />
</LinearLayout>
```

在布局文件中将 EditText 的输入类型改为“phone”，其他的都是常规的属性。

运行程序，在输入框内输入要拨打的电话号码，然后点击“拨打电话”按钮，并不会直接拨打电话，而是进入拨号界面，如图 11-1 所示。

再点击“拨打电话”按钮才真正开始拨号，如图 11-2 所示。

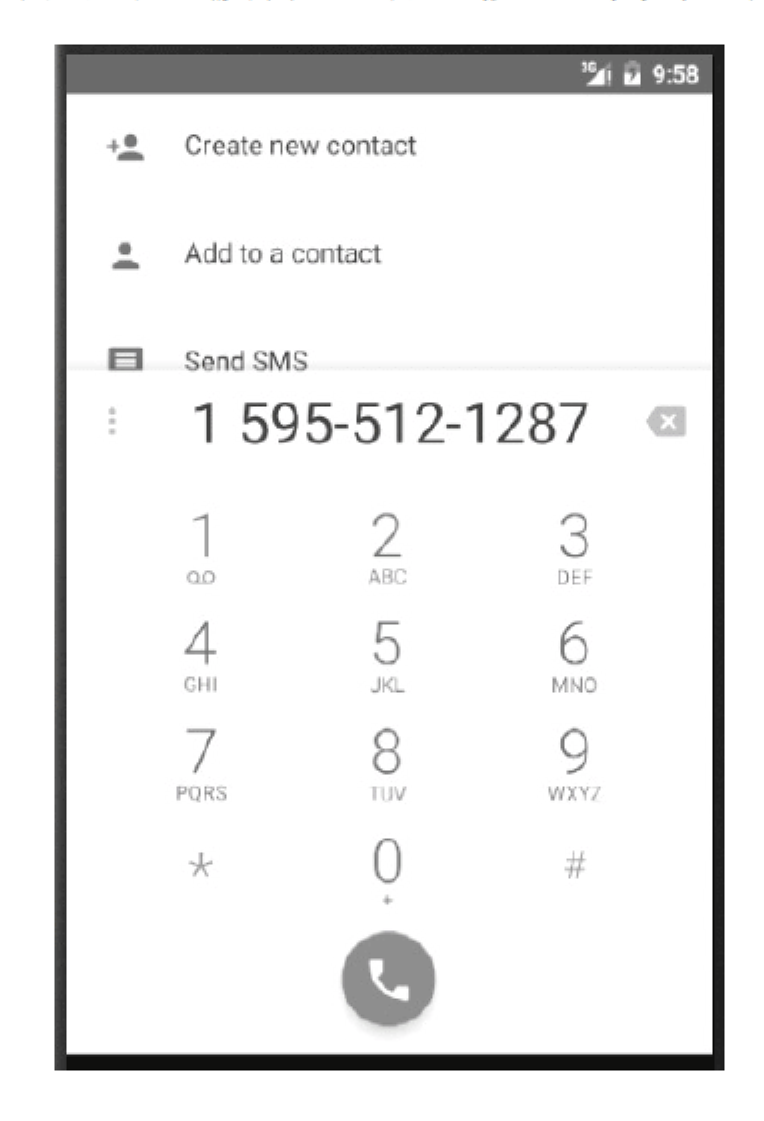


图 11-1 使用“Intent.ACTION_DIAL”进行拨号（1）

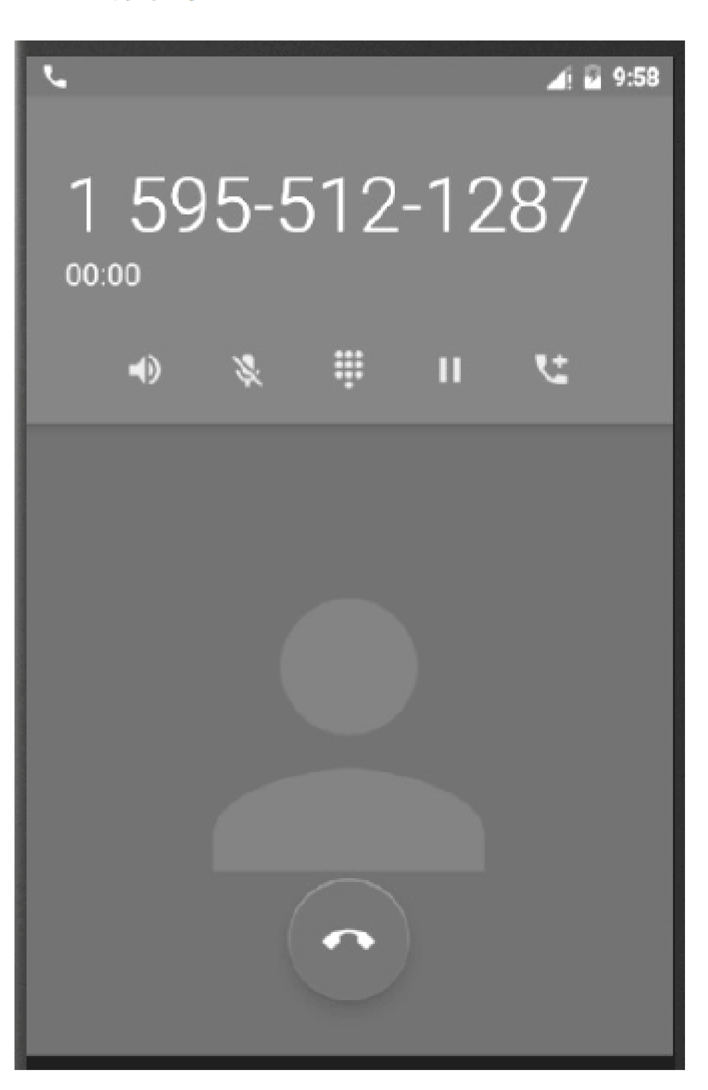


图 11-2 使用“Intent.ACTION_DIAL”进行拨号（2）

到这里可能有读者会疑惑，程序中没有添加关于拨号权限的声明，为什么可以正常运行？其实从上面的运行结果就可以看出原因，我们的应用并没有直接去进行拨号操作，而是先调换到拨号界面，由系统的拨号应用进行拨号操作，所以这里并不需要权限声明。如果将上述代码中的 Intent 部分改为下面这种方式：

```
//Intent callIntent = new Intent(Intent.ACTION_DIAL);
Intent callIntent = new Intent(Intent.ACTION_CALL);
```

运行程序，就会发现提示没有相关权限。因为这种方式会直接进行拨号，所以需要权限。又由于拨号是危险权限，因此必须进行动态获取。此时将 MainActivity 代码修改如下：

```
package com.buaa.call;

import android.Manifest;
import android.content.Intent;
import android.content.pm.PackageManager;
import android.net.Uri;
import android.os.Build;
import android.support.v4.app.ActivityCompat;
import android.support.v4.content.ContextCompat;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
```

```
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {

    private EditText number;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        initView();
    }

    private void initView() {
        number = (EditText) findViewById(R.id.phone_number);
        Button button = (Button) findViewById(R.id.call_but);
        button.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                //Intent callIntent = new Intent(Intent.ACTION_DIAL);
                getPermission();
            }
        });
    }

    private void callPhone() {
        Intent callIntent = new Intent(Intent.ACTION_CALL);
        //这里向拨号器传递数据，在号码前面必须加上"tel:"
        Uri data = Uri.parse("tel:" + number.getText().toString());
        callIntent.setData(data);
        startActivity(callIntent);
    }

    public void getPermission() {
        //判断版本号，在 api23 也就是 6.0 版本之前能直接获得权限
        if (Build.VERSION.SDK_INT >= 23) {
            int checkCALLPermission = ContextCompat.
                    checkSelfPermission(this,
                            Manifest.permission.CALL_PHONE);
            //判断是否具有权限
            if (checkCALLPermission != PackageManager.PERMISSION_GRANTED) {
```

```
                //用以申请权限的方法，此时使用 ActivityCompat 类的方法，以便于版本兼容
                ActivityCompat.requestPermissions(this,
                        new String[]{Manifest.permission.CALL_PHONE},
                        1);
                return;
            } else {
                //如果已经获取了相关权限，调用 initData()与 initView()方法
                callPhone();
            }
        } else {
            //如果 api 版本低于 23，就直接调用 initData()与 initView()方法
            callPhone();
        }
    }

    //申请权限做出响应后的回调函数
    @Override
    public void onRequestPermissionsResult(
            int requestCode, String[] permissions, int[] grantResults) {
        switch (requestCode) {
            case 1:
                if (grantResults[0] == PackageManager.PERMISSION_GRANTED) {
                    Toast.makeText(this, "获取权限成功", Toast.LENGTH_SHORT)
                            .show();
                    //获取权限成功，拨打电话
                   callPhone();
                } else {
                    Toast.makeText(this, "获取权限失败", Toast.LENGTH_SHORT)
                            .show();
                }
                break;
            default:
                super.onRequestPermissionsResult(
                        requestCode, permissions, grantResults);
        }
    }
}
```

同时在 AndroidManifest.xml 中注册权限：

```
<uses-permission android:name="android.permission.CALL_PHONE"/>
```

运行程序，在编辑框中输入号码后点击“拨打电话”按钮，会先提示用户是否允许应用拨打电话，如图 11-3 所示。

如果用户点击“ALLOW”就会直接进行拨号，如图 11-4 所示。



图 11-3　提示是否允许拨打电话

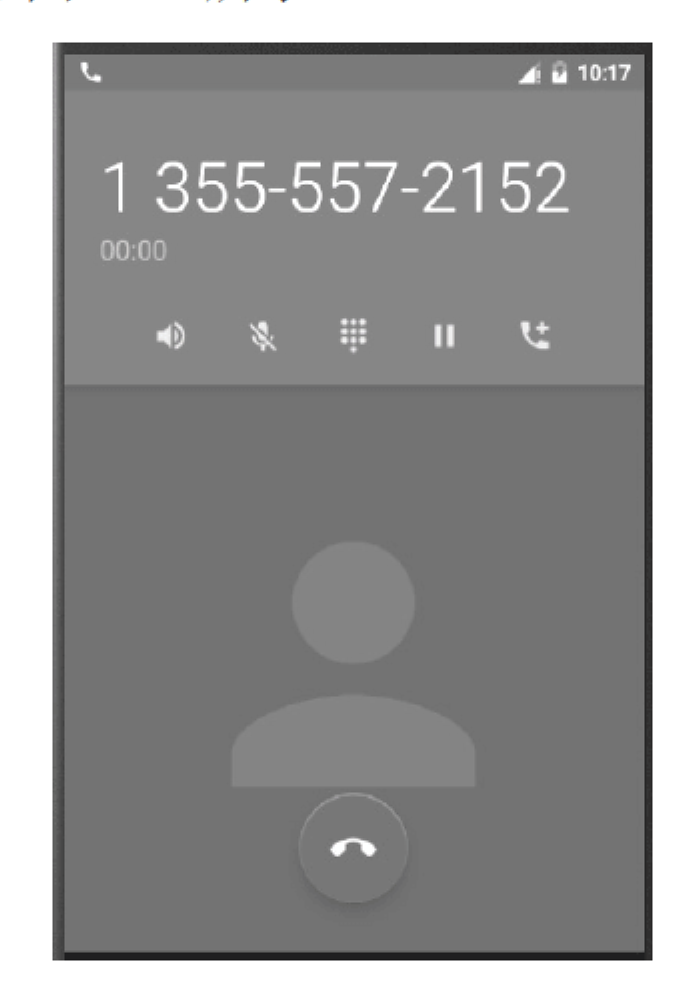


图 11-4　使用“Intent.ACTION_CALL”直接拨号

本部分讲述了两种拨号方式，它们各有优劣，需要在应用中根据业务场景的不同来进行选择。

11.1.2　短信发送

在 Android 开发中，常常有在应用中发送短信的需求，比如在支付宝中的发送查询信用卡还款额度的短信等。在 Android 开发中，发送短信有两种方式，第一种方式是通过 startActivity(Intent intent) 来打开短信应用，其中的参数是一个 aciton 为“Intent.ACTION_SENDTO”的 Intent；第二种方式是使用 SmsManager 管理器来发送短信。其中，前者相对简单，而且与 11.1.1 小节拨号的第一种方式一样，并不需要获取权限，这是因为它只是开启了短信应用并将相关数据传入，但是没有执行发送的操作，真正的发送短信还是在短信管理器中进行的。后一种方式会直接在当前应用中发送短信，同时还可以对发送状态进行监听。

下面通过实例来进行说明。新建一个 sendSms 项目，修改 activity_main.xml 布局文件：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context="com.buaa.sms.MainActivity">

    <EditText
        android:id="@+id/sms_content"
        android:layout_width="match_parent"
        android:layout_height="200dp"
```

```
            android:layout_marginTop="20dp"
            android:gravity="top|left"
            android:hint="输入要发送的内容"
            android:singleLine="false"
            android:textSize="22sp" />

        <EditText
            android:id="@+id/phone_number"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_marginTop="20dp"
            android:hint="输入号码"
            android:inputType="phone"
            android:singleLine="true"
            android:textSize="22sp" />

        <Button
            android:id="@+id/sms_button"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_marginTop="20dp"
            android:text="发送" />
    </LinearLayout>
```

这里使用了两个 EditText，用于输入短信内容和要发送的号码；以及一个 Button 按钮，用于进行点击事件。下面在 MainActivity 中获取这些控件并进行相关操作，代码如下：

```
    public class MainActivity extends AppCompatActivity {

        private EditText smsNumber;
        private EditText smsContent;

        @Override
        protected void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            setContentView(R.layout.activity_main);
            initView();
        }

        private void initView() {
            smsContent = (EditText) findViewById(R.id.sms_content);
            smsNumber = (EditText) findViewById(R.id.phone_number);
            Button smsButton = (Button) findViewById(R.id.sms_button);
            smsButton.setOnClickListener(new View.OnClickListener() {
```

```
                @Override
                public void onClick(View v) {
                    sendSmsByIntent();
                }
            });
        }

        private void sendSmsByIntent() {
            Intent intent = new Intent(Intent.ACTION_SENDTO);
            //固定格式，必须是"sms_body"
            intent.putExtra("sms_body", smsContent.getText().toString());
            //固定格式，必须是"smsto"
            Uri data = Uri.parse("smsto:" + smsNumber.getText().toString());
            intent.setData(data);
            startActivity(intent);
        }
    }
```

这里并不需要加入权限，直接运行程序即可。输入短信内容和号码，点击按钮之后会跳转到短信管理器中，效果如图 11-5 中的左图所示。此时点击发送就会出现图 11-5 中右侧的界面，表明短信确实已被发送出去。

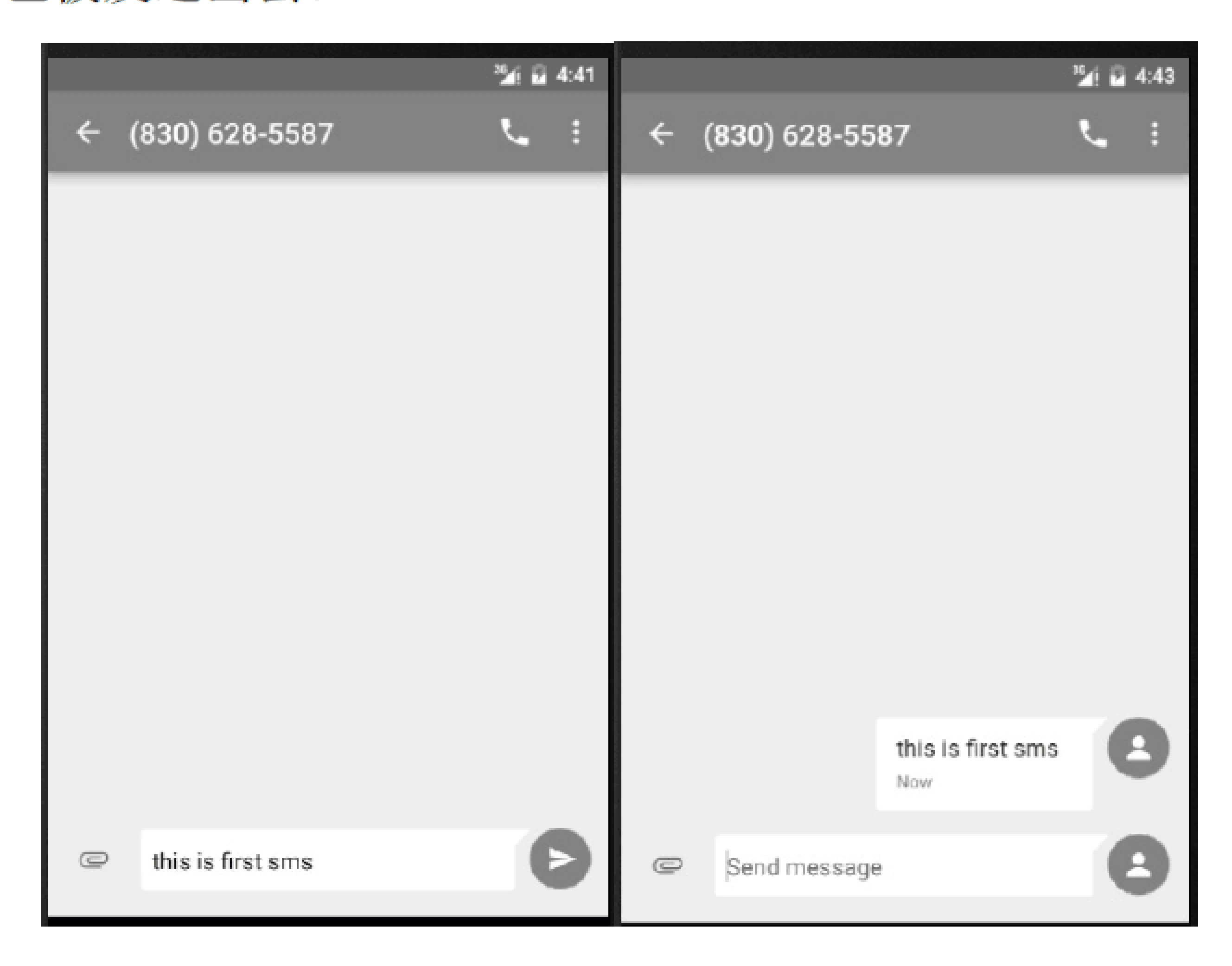


图 11-5　通过 startActivity 方法发送短信

以上面这种方式发送短信相对简单，但其实并不是真在我们自己的应用中发送短信。如果想要在自己的应用中发送短信，就需要借助 SmsManager 来获取短信管理器，然后进行短信的发送。通过 SmsManager 方式来发送短信，可以选择对发送状态和对方的接收状态进行监听，当然也可以选择不做监听。监听使用的技术是广播。另外，此时是在应用内发送短信，因此需要申请相关权限，而短信相关权限属于危险权限，因此必须动态申请。下面通过对 MainActivity 做修改来进行说明，代码如下：

```
public class MainActivity extends AppCompatActivity {

    private EditText smsNumber;
    private EditText smsContent;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        initView();
    }

    private void initView() {
        smsContent = (EditText) findViewById(R.id.sms_content);
        smsNumber = (EditText) findViewById(R.id.phone_number);
        Button smsButton = (Button) findViewById(R.id.sms_button);
        smsButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                getPermission();
            }
        });
    }

    private void sendSmsBySmsManager() {
        String SENT_SMS_ACTION = "SENT_SMS_ACTION";
        Intent sentIntent = new Intent("SENT_SMS_ACTION");
        //对发送状态进行监听
        PendingIntent sentPI = PendingIntent.getBroadcast(this, 0, sentIntent, 0);

        String DELIVERED_SMS_ACTION = "DELIVERED_SMS_ACTION";
        Intent deliverIntent = new Intent(DELIVERED_SMS_ACTION);
        //对对方的接收状态进行监听
        PendingIntent deliverPI = PendingIntent.getBroadcast(this, 0,
                deliverIntent, 0);

        //获取短信管理器
        SmsManager smsManager = SmsManager.getDefault();
        //拆分短信内容（手机短信长度限制）
        List<String> divideContents = smsManager.divideMessage(smsContent.getText().toString());
        for (String text : divideContents) {
            smsManager.sendTextMessage(smsNumber.getText().toString(), null, text, sentPI, deliverPI);
        }
```

```
        //对发送状况进行监听
        registerReceiver(new BroadcastReceiver() {
            @Override
            public void onReceive(Context context, Intent intent) {
                switch (getResultCode()) {
                    case Activity.RESULT_OK:
                        Toast.makeText(context,
                                "短信发送成功", Toast.LENGTH_SHORT)
                                .show();
                        break;
                    case SmsManager.RESULT_ERROR_GENERIC_FAILURE:
                        Toast.makeText(context,
                                "短信发送失败", Toast.LENGTH_SHORT)
                                .show();
                        break;
                    case SmsManager.RESULT_ERROR_RADIO_OFF:
                        Toast.makeText(context,
                                "短信发送失败", Toast.LENGTH_SHORT)
                                .show();
                        break;
                    case SmsManager.RESULT_ERROR_NULL_PDU:
                        Toast.makeText(context,
                                "短信发送失败", Toast.LENGTH_SHORT)
                                .show();
                        break;
                }
            }
        }, new IntentFilter(SENT_SMS_ACTION));

        //对对方接收状况进行监听
        this.registerReceiver(new BroadcastReceiver() {
            @Override
            public void onReceive(Context context, Intent intent) {
                Toast.makeText(context,
                        "收信人已经成功接收", Toast.LENGTH_SHORT)
                        .show();
            }
        }, new IntentFilter(DELIVERED_SMS_ACTION));
    }

    public void getPermission() {
        //判断版本号，在 api23 也就是 6.0 版本之前能直接获得权限
```

```
            if (Build.VERSION.SDK_INT >= 23) {
                int checkCALLPermission = ContextCompat.
                        checkSelfPermission(this,
                                Manifest.permission.SEND_SMS);
                //判断是否具有权限
                if (checkCALLPermission != PackageManager.PERMISSION_GRANTED) {
                    //用以申请权限的方法，此时使用 ActivityCompat 类的该方法，以便于版本兼容
                    ActivityCompat.requestPermissions(this,
                            new String[]{Manifest.permission.SEND_SMS},
                            1);
                    return;
                } else {
                    sendSmsBySmsManager();
                }
            } else {
                sendSmsBySmsManager();
            }
        }

        //申请权限做出响应后的回调函数
        @Override
        public void onRequestPermissionsResult(
                int requestCode, String[] permissions, int[] grantResults) {
            switch (requestCode) {
                case 1:
                    if (grantResults[0] == PackageManager.PERMISSION_GRANTED) {
                        Toast.makeText(this, "获取权限成功", Toast.LENGTH_SHORT)
                                .show();
                        //获取权限成功，发送短信
                        sendSmsBySmsManager();
                    } else {
                        Toast.makeText(this, "获取权限失败", Toast.LENGTH_SHORT)
                                .show();
                    }
                    break;
                default:
                    super.onRequestPermissionsResult(
                            requestCode, permissions, grantResults);
            }
        }
    }
```

在上面的代码中，发送短信主要依靠 sendSmsBySmsManager()方法，其他的主要是用于获

取控件和获取权限的方法。在 sendSmsBySmsManager()方法中，我们通过调用 SmsManager 的 getDefault()方法获取 SmsManager 的实例，然后调用 sendTextMessage()方法就可以发送短信了。sendTextMessage()方法接收 5 个参数，其中第一个参数用于指定接收人的手机号码，第三个参数用于指定短信的内容，其他几个参数可以为 null。另外，根据国际标准，每条短信的长度不得超过 160 个字符，如果想要发送超出这个长度的短信，就需要将这条短信分割成多条短信来发送。

如果想要对短信发送状况以及接收方的接收状况进行监听，就必须用 PendingIntent 的 getBroadcast()方法获取一个 PendingIntent 对象，并将其作为第四个参数传递到 sendTextMessage()方法中，然后动态注册广播接收者来监听短信的发送状态。同样的道理，如果想要接收对方的接收状况，就必须用 PendingIntent 的 getBroadcast()方法获取一个 PendingIntent 对象，并将其作为第五个参数传递到 sendTextMessage()方法中，然后动态注册广播接收者来监听对方接收短信的状态。

完成这些之后，向 AndroidManifest.xml 中加入权限声明：

```
<uses-permission
android:name="android.permission.SEND_SMS"/>
```

运行程序，在输入框中输入短信内容和手机号之后点击“发送”，系统会先提示是否授予应用短信发送的权限，此时点击“ALLOW”，获取到权限之后短信将被发送出去，并出现一个 Toast，提示短信发送成功，如图 11-6 所示。

由于这里使用的是模拟器，短信其实并没有被发送，所以无法接收到对方接收短信的状况。有兴趣的读者可以使用真机进行验证。

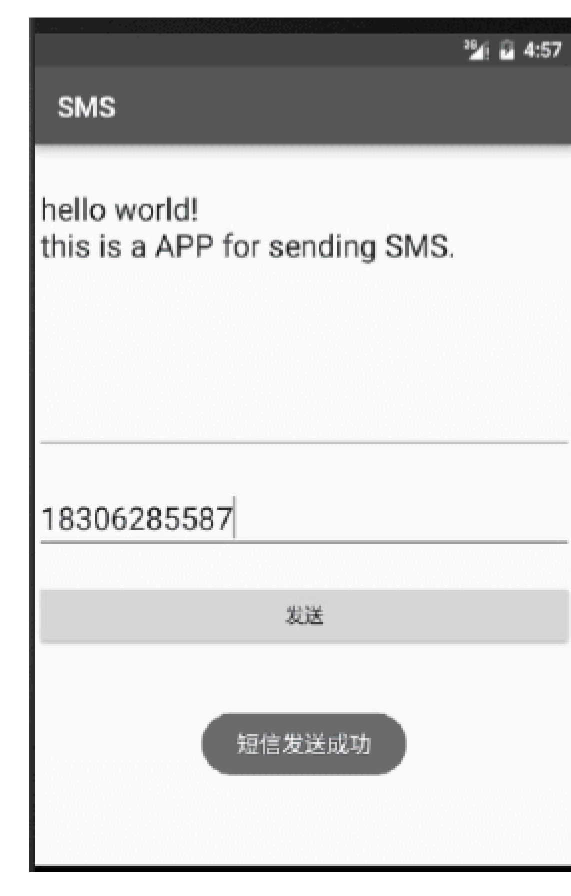


图 11-6　借助 SmsManager 发送短信

11.1.3　接收短信

在 Android 应用中短信的发送与接收都经常使用，其中短信接收使用频率更高。一个典型的场景是应用中需要进行短信验证时，用户需根据系统发送短信中的验证码进行验证，这时如果应用能够直接获取验证码并填写进输入框中就会有较好的用户体验。

【接收短信实例】

下面通过一个实例来进行分析。创建一个项目 receive_sms，修改 activit_main.xml 文件：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context="com.buaa.receive_sms.MainActivity">
```

```
    <EditText
        android:id="@+id/confirm_number"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:paddingTop="40dp"
        android:textSize="22sp" />

    <EditText
        android:id="@+id/confirm_text"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="20dp"
        android:textSize="22sp" />

    <Button
        android:id="@+id/confirm_button"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="20dp"
        android:text="获取短信验证码" />
</LinearLayout>
```

这里使用了两个 EditText，分别用于展示短信内容和发送方号码，并用一个 Button 按钮来开启广播接收者。下面通过修改 MainActivity 来获取控件并设置 Button 的点击事件，代码如下：

```
public class MainActivity extends AppCompatActivity {

    private SmsReceiver smsReceiver;
    private EditText smsContent;
    private EditText smsNumber;
    private Handler handler = new Handler() {
        @Override
        public void handleMessage(Message msg) {
            super.handleMessage(msg);
            if (msg.what == 123) {
                Bundle bundle = msg.getData();
                smsContent.setText(bundle.getString("content"));
                smsNumber.setText(bundle.getString("number"));
                unregisterReceiver(smsReceiver);
            }
        }
    };
```

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    initView();
}

private void initView() {
    smsContent = (EditText) findViewById(R.id.confirm_text);
    smsNumber = (EditText) findViewById(R.id.confirm_number);
    Button smsButton = (Button) findViewById(R.id.confirm_button);
    smsButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            getPermission();
        }
    });
}

private void registerSmsReceiver() {
    IntentFilter receiveFilter = new IntentFilter();
    receiveFilter.addAction("android.provider.Telephony.SMS_RECEIVED");
    smsReceiver = new SmsReceiver(handler);
    registerReceiver(smsReceiver, receiveFilter);
}

public void getPermission() {
    //判断版本号，在 api23 也就是 6.0 版本之前能直接获得权限
    if (Build.VERSION.SDK_INT >= 23) {
        int checkCALLPermission = ContextCompat.
                checkSelfPermission(this,
                        Manifest.permission.RECEIVE_SMS);
        //判断是否具有权限
        if (checkCALLPermission != PackageManager.PERMISSION_GRANTED) {
            //用以申请权限的方法，此时使用 ActivityCompat 类中的该方法，以便于版本兼容
            ActivityCompat.requestPermissions(this,
                    new String[]{Manifest.permission.RECEIVE_SMS},
                    1);
            return;
        } else {
            registerSmsReceiver();
        }
```

```
            } else {
                registerSmsReceiver();
            }
        }

        //申请权限做出响应后的回调函数
        @Override
        public void onRequestPermissionsResult(
                int requestCode, String[] permissions, int[] grantResults) {
            switch (requestCode) {
                case 1:
                    if (grantResults[0] == PackageManager.PERMISSION_GRANTED) {
                        Toast.makeText(this, "获取权限成功", Toast.LENGTH_SHORT)
                                .show();
                        registerSmsReceiver();
                    } else {
                        Toast.makeText(this, "获取权限失败", Toast.LENGTH_SHORT)
                                .show();
                    }
                    break;
                default:
                    super.onRequestPermissionsResult(
                            requestCode, permissions, grantResults);
            }
        }
    }
```

在 Activity 类中设置了 Button 的点击事件，当触发时会开启广播接收者，我们会在广播接收者中对短信进行处理，并通过 Handler 机制将接收到的内容更新到 UI。又由于接收短信属于危险权限，因此需要动态获取。不管是事件处理、Handler 机制、开启广播接收者还是动态获取权限都已经接触过很多次，这里不再解释。此外，本实例中在更新完 UI 后才关闭广播接收者，这是因为我们只想在需要获取验证码时开启广播接收者对广播进行监听，获取之后希望将其关闭。另外，这里需要新建一个广播接收者类 SmsReceiver 来处理具体的操作，代码如下：

```
    public class SmsReceiver extends BroadcastReceiver {
        private Handler handler;

        public SmsReceiver(Handler handler) {
            this.handler = handler;
        }

        @Override
        public void onReceive(Context context, Intent intent) {
```

```
            //获取短信类型
            String format = intent.getStringExtra("format");
            Bundle bundle = intent.getExtras();
            // 提取短信消息
            Object[] pdus = (Object[]) bundle.get("pdus");
            SmsMessage[] messages = new SmsMessage[pdus.length];

            if (Build.VERSION.SDK_INT >= 23) {
                for (int i = 0; i < messages.length; i++) {
                    //此方法适用于 Android 6.0 以上版本
                    messages[i] = SmsMessage.createFromPdu((byte[]) pdus[i], format);
                }
            } else {
                //此方法在 Android 6.0 版本时被废弃
                for (int i = 0; i < messages.length; i++) {
                   messages[i] = SmsMessage.createFromPdu((byte[]) pdus[i]);
                }
            }
            String number = messages[0].getOriginatingAddress();
            String content = "";
            for(SmsMessage message:messages){
                //获取短信内容
                content+=message.getMessageBody();
            }

            Message message=handler.obtainMessage();
            message.what=123;
            Bundle bundle1=new Bundle();
            bundle.putString("number",number);
            bundle.putString("content",content);
            message.setData(bundle);
            handler.sendMessage(message);
        }
    }
```

SmsReceiver 继承了 BroadcastReceiver 类，实现了 onReceive()方法，并通过构造方法对 Handler 进行了初始化。在 onReceive()方法中，首先我们从 Intent 参数中取出一个 Bundle 对象，然后使用 pdu 密钥来提取一个 SMS pdus 数组，其中每一个 pdu 都表示一条短信消息。接着使用 SmsMessage 的 createFromPdu()方法将每一个 pdu 字节数组转换为 SmsMessage 对象，调用这个对象的 getOriginatingAddress()方法就可以获取短信的发送方号码，调用 getMessageBody()方法获取短信的内容，然后将每一个 SmsMessage 对象中的短信内容拼接起来，就组成了一条完整的短信。最后将获取到的发送方号码和短信内容通过 Handler 发送到主线程，显示在 EditText 中。在这里需要强调一下，createFromPdu()方法在 Android 6.0 版本之

后发生了变化。在 6.0 版本之前使用的该方法只需要传入一个字节数组类型的参数，但是在 6.0 版本之后这种只传入一个参数的方法被废弃了，除了字节数组还需要传递一个代表短信类型的字符串外，这个字符串可以通过 intent.getStringExtra("format")来获取。所以在将 pdu 字节数组转换为 SmsMessage 对象时需要先判断版本号再具体使用不同的方法。

完成这些内容之后，只需要在 AndroidManifest.xml 文件中加上权限声明即可：

```
<uses-permission android:name="android.permission.RECEIVE_SMS" />
```

运行程序，点击“获取短信验证码”注册广播接收者。这时如果发送一条短信到本机，应用就能够捕获到该条短信，并将内容和发送号码显示到要输入的地方，如图 11-7 所示。

由于模拟器的局限性，在发送短信时并不能够真正发出，所以这里使用真机来进行操作（号码进行了涂改）。



图 11-7　接收短信的实现

11.2　再论 Notification

Notification 俗称通知，是一种具有全局效果的通知，展示在屏幕的顶端，首先会表现为一个图标的形式，当用户向下滑动的时候，展示通知的具体内容。通知在 Android 中的运用是很常见的，可以让我们在获得消息的时候在状态栏、锁屏界面显示相应的信息，很难想象如果没有通知机制，qq、微信以及其他应用将如何通知用户。本节会介绍 3 种 Notification，分别是普通 Notification、折叠式 Notification 和悬挂式 Notification。另外，在 Notification 中，还有一种大视图通知，与一般的通知在使用上并无过多区别，本节不做讲解，如果有读者对此感兴趣可配合文档进行研究。

11.2.1　普通 Notification 回顾与拓展

在第 8 章讲解前台服务时已经对 Notification 做了简单的讲解，这里做一下简单的回顾。

创建一个 Notification 时，首先需要一个 NotificationManager 来对 Notification 进行管理。NotificationManager 是一个重要的系统级服务，位于应用程序的框架层中，应用程序可以通过它向系统发送全局通知。这个对象是由系统维护的服务，以单例模式获得，所以一般并不直接实例化这个对象，而是通过调用 Context 的 getSystemService()方法获取。getSystemService()方法接收一个字符串参数，用于确定获取系统的哪个服务，这里我们传入 Context.NOTIFICATION_SERVICE 即可。因此，获取 NotificationManager 的实例就可以写成：

```
NotificationManager notificationManager = (NotificationManager)
getSystemService(NOTIFICATION_SERVICE);
```

除了 NotificationManager 之外，还需要创建一个 Notification 对象。考虑版本兼容性的问题，创建一个 Notification 对象需要分三种情况。第一种是 API 版本低于 11 的，只能使用

Notification 的 setLatestEventInfo()方法。由于现在很少有手机支持 API 11 以下的版本了，因此本机不做详细介绍，有兴趣的读者可以自行阅读官方文档。第二种是 API 版本高于 11 并低于 16（Android 4.1.2）的，可使用 NotificationCompat.Builder 来构造 Notification，但要使用 getNotification()来获取 Notification 对象。第三种情况是 API 高于 16 的，可以用 Builder 和 build()函数来配套地使用 Notification。

当 NotificationManager 和 Notification 都创建之后，就可以使用 notificationManager.notify(1, notification)方法来开启通知了。简单示例如下：

```
private void normalNotification() {
    Intent intent = new Intent(this, SecondActivity.class);
    PendingIntent pendingIntent = PendingIntent.getActivity(this, 0, intent,
PendingIntent.FLAG_CANCEL_CURREN);

    Notification notification;
    NotificationCompat.Builder builder = new NotificationCompat.Builder(this);
    //设置属性
    ......
    if (Build.VERSION.SDK_INT >= 16) {
        notification = builder.build();
    } else {
        notification = builder.getNotification();
    }
    NotificationManager notificationManager =
            (NotificationManager) getSystemService(NOTIFICATION_SERVICE);
    notificationManager.notify(0, notification);
}
```

通过这个示例，可以看出这里一共分 3 个部分：第一部分创建一个 PendingIntent，第二个部分针对不同版本创建 Notification 对象，并设置了一些属性；第三部分创建了一个 NotificationManager 对象，并用它开启了通知。读者可能对 PendingIntent 并不熟悉，同时对 Notification 的一些属性也缺乏足够的了解，下面将分别对它们进行介绍。

1. PendingIntent

PendingIntent 是一种特殊的 Intent，主要区别在于 Intent 的执行是立刻的，而 PendingIntent 的执行不是立刻的。PendingIntent 执行的操作实质上是参数传进来的 Intent 操作，但是使用 PendingIntent 的目的在于它所包含的 Intent 操作的执行是需要满足某些条件的。主要使用示例包括通知 Notification 的发送、短消息 SmsManager 的发送和警报器 AlarmManager 的执行等。

PendingIntent 可以看作是对 Intent 的包装，通常通过 getActivity()、getBroadcast()、getService()来得到 PendingIntent 的实例。当前 Context 并不能马上启动它所包含的 Intent，而是在外部执行 PendingIntent 时调用 Intent。正由于 PendingIntent 中保存有当前 App 的 Context，使其赋予外部 App 一种能力，即外部 App 可以如同当前 App 一样执行 PendingIntent 里的 Intent，就算在执行时当前 App 已经不存在了，也能通过存在 PendingIntent 里的 Context 执行 Intent。

要得到一个PendingIntent对象，可以使用PendingIntent类的静态方法：getActivity(Context context, int requestCode,Intent intent, @Flags int flags)、getBroadcast(Context context, int requestCode,Intent intent, @Flags int flags)和 getService(Context context, int requestCode,Intent intent, @Flags int flags)。这 3 种方法需要传入的参数是一致的，都需要 4 个：第一个是当前的上下文；第二个是请求码；第三个是一个 Intent 对象，分别对应着 Intent 的 3 个行为，即跳转到一个 activity 组件、打开一个广播组件和打开一个服务组件；第四个参数用于确定 PendingIntent 的行为，有 FLAG_ONE_SHOT、FLAG_NO_CREATE、FLAG_CANCEL_CURRENT 和 FLAG_UPDATE_CURRENT 这 4 种值可选，每种值的含义可由文档查看。这里特别说明一下第二个参数，在很多书籍对此并没有加以重视，甚至说只需填写为 0 即可，这显然是不负责任的。当有多个 PendingIntent 消失时，第二个参数 requestCode 的作用就会显现，如果此时 requestCode 值一样，后面的 PendingIntent 就会对之前的消息起作用，所以为了避免影响之前的消息，requestCode 每次都要设置不同的内容。

2. Notification 的属性

Notification 的属性很多，使用 set 方法设置即可。表 11-1 给出了一些常用的 set 方法。

表11-1　Notification中常用的set方法

Set 方法	作用介绍
public Builder setTicker(CharSequence tickerText)	设置状态栏开始动画的文字，可选
public Builder setContentTitle(CharSequence title)	设置内容区的标题，必须设置
public Builder setContentText(CharSequence text)	设置内容区的内容，必须设置
public Builder setColor(@ColorInt int argb)	设置 smallIcon 的背景色，可选
public Builder setSmallIcon(@DrawableRes int icon)	设置小图标，必须设置
public Builder setLargeIcon(Bitmap b)	设置打开通知栏后的大图标
public Builder setWhen(long when)	设置显示通知的时间，不设置默认获取系统时间，会在 Notification 中显示
public Builder setAutoCancel(boolean autoCancel)	设置为 true，点击该条通知会自动删除，设置为 false 时只能通过滑动来删除
public Builder setPriority(int pri)	设置优先级，级别高的排在前面
public Builder setDefaults(int defaults)	设置上述铃声，振动、闪烁用|分隔，常量在 Notification 里
public Builder setOngoing(boolean ongoing)	设置是否为一个正在进行中的通知，这一类型的通知将无法删除
public Builder setContentIntent(PendingIntent intent)	设置点击通知时执行的任务
public Builder setVisibility(int visibility)	设置 Notification 的显示等级，共有 3 种：VISIBILITY_PUBLIC 表示只有在没有锁屏时会显示通知；VISIBILITY_PRIVATE 表示任何情况都会显示通知；VISIBILITY_SECRET 表示在安全锁和没有锁屏的情况下显示通知

下面根据上述属性来演示设置的效果。创建一个新的工程 notification，修改 MainActivity：

```
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        normalNotification("hello", "普通通知", "hello world", true, true, true);
    }

    private void normalNotification(String ticker, String title, String content, boolean sound, boolean vibrate,
        boolean lights) {
        Intent intent = new Intent(this, SecondActivity.class);
        PendingIntent pendingIntent = PendingIntent.getActivity(this, 0, intent,
                                PendingIntent.FLAG_CANCEL_CURRENT);
        Notification notification;
        NotificationCompat.Builder builder = new NotificationCompat.Builder(this);
        builder.setContentIntent(pendingIntent);
        builder.setAutoCancel(true);
        builder.setTicker(ticker);
        builder.setContentTitle(title);
        builder.setContentText(content);
        builder.setColor(Color.RED);
        builder.setSmallIcon(R.mipmap.ic_launcher);
        builder.setLargeIcon(BitmapFactory.decodeResource(getResources(), R.mipmap.ic_launcher));
        builder.setWhen(System.currentTimeMillis());
        builder.setAutoCancel(true);
        builder.setPriority(NotificationCompat.PRIORITY_MAX);
        int defaults = 0;
        if (sound) {
            defaults |= Notification.DEFAULT_SOUND;
        }
        if (vibrate) {
            defaults |= Notification.DEFAULT_VIBRATE;
        }
        if (lights) {
            defaults |= Notification.DEFAULT_LIGHTS;
        }
        builder.setDefaults(defaults);
        builder.setOngoing(true);
        if (Build.VERSION.SDK_INT >= 16) {
            notification = builder.build();
        } else {
            notification = builder.getNotification();
```

```
            }
            NotificationManager notificationManager =
                    (NotificationManager) getSystemService(NOTIFICATION_SERVICE);
            notificationManager.notify(0, notification);
        }
    }
```

直接运行程序，由于程序中是在 onCreate()方法调用此通知，因此程序刚一运行就会听到一声提示音，如果使用真机测试还会伴随着震动。在状态栏的效果如图 11-8 所示。



图 11-8 通知的基本使用

标题、内容、图标、时间都和预期的一致，此时点击当前通知就会执行 PendingIntent，然后进入 SecondActivity，这里不再演示。另外，本实例中设置的是 builder.setAutoCancel(true)，即点击该通知，通知消失，如果设置为 false，点击通知就不会主动消失（除非在通知栏滑动它），需要在点击之后跳转进入的 context 中取消通知，只需使用 getSystemService(NOTIFICATION_SERVICE)获取 NotificationManager 类的实例，然后，调用 notificationManager.cancel(0)即可。当然，cancel()方法中的参数要与 notificationManager.notify(0, notification)中的保持一致。

11.2.2 折叠式 Notification

折叠式 Notification 是一种自定义视图的 Notification，用来显示长文本和一些自定义的布局场景。它有两种状态，一种是普通状态下的视图，但是这种状态和上面普通通知的视图样式一样；一种是展开状态下的视图。和普通 Notification 不同的是，它需要自定义视图，而这个视图显示的进程和创建视图的进程不在一个进程，所以需要使用 RemoteViews 来创建自定义视图。

在上一部分实例的基础上，先创建一个布局文件 remote_view.xml，代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="200dp"
    android:orientation="horizontal">

    <ImageView
        android:layout_width="100dp"
        android:layout_height="100dp"
        android:layout_gravity="center"
        android:src="@drawable/ic_launcher" />

    <TextView
        android:layout_width="wrap_content"
```

```
            android:layout_height="wrap_content"
            android:layout_gravity="center_vertical"
            android:layout_marginLeft="100dp"
            android:text="自定义通知中的折叠部分"
            android:textColor="@color/colorPrimaryDark"
            android:textSize="22sp" />
</LinearLayout>
```

然后在上一实例的 normalNotification()方法中加入下面两行代码：

```
RemoteViews remoteViews = new RemoteViews(getPackageName(), R.layout.remote_view);
notification.bigContentView = remoteViews;
```

重新运行程序，除了上例中的效果外，在状态栏中下拉该通知时会出现如图 11-9 所示的效果。



图 11-9　折叠式通知

11.2.3　悬挂式 Notification

悬挂式 Notification 是 Android 5.0 新增加的方式，和前两种显示方式不同的是，前两种需要下拉通知栏才能看到通知，而悬挂式 Notification 不需要下拉通知栏就直接显示出来悬挂在屏幕上方，并且焦点不变，仍在用户操作的界面，因此不会打断用户的操作，过几秒就会自动消失。实现悬挂式 Notification 需要调用 setFullScreenIntent()方法来将普通 Notification 变为悬挂式 Notification。具体来说，只需要在上例的基础上加入 builder.setFullScreenIntent(pendingIntent,true) 即可。运行程序，效果如图 11-10 所示。

此时，焦点仍在当前的 Activity，如果不对此通知做任何操作，很快它就消失了。

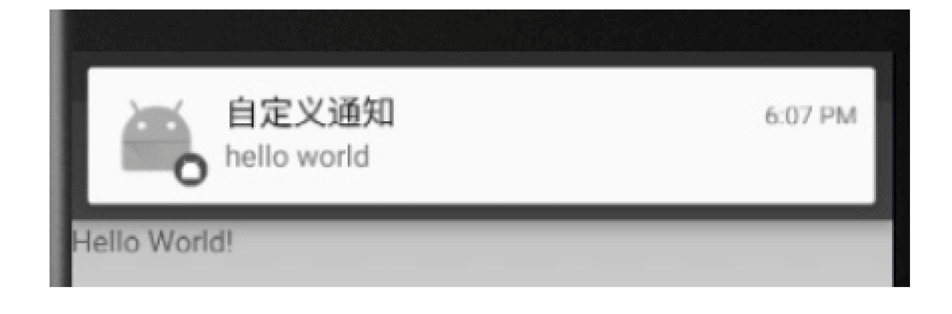


图 11-10　悬挂式通知

11.2.4　Notification 的其他应用

在开发过程中经常还会有两种常见的需求，一种是当应用在状态栏显示一条通知时，用户看到并希望直接在通知处进行操作；另一种是当下载时希望使用进度条进行显示。下面分别介绍两者的实现。

1. 添加可点击按钮的通知

添加可点击按钮的方法很简单，只需要调用 builder 的 addAction(Action action)方法即可。

该方法中的参数为 NotificationCompat.Action 类型，当然，此处 Notification.Action 类型也是支持的，只不过官方已经不再推荐使用。获取 Action 类对象，需要使用该类带参数的构造方法。该类构造方法的参数有 3 个，第一个为显示的图片，第二个为显示的文本，第三个为一个 PendingIntent 类型的参数，是点击之后要执行的动作。

在实例中，只需要将 normalNotification()方法的代码修改如下即可：

```
private void normalNotification(String title, String content) {
    Intent intent = new Intent(this, SecondActivity.class);
    PendingIntent pendingIntent = PendingIntent.getActivity(this, 0, intent,
PendingIntent.FLAG_UPDATE_CURRENT);
    Notification notification;
    NotificationCompat.Builder builder = new NotificationCompat.Builder(this);
    builder.setContentIntent(pendingIntent);

    builder.setAutoCancel(true);
    builder.setContentTitle(title);
    builder.setContentText(content);
    builder.setSmallIcon(R.mipmap.ic_launcher);
    builder.setLargeIcon(BitmapFactory.decodeResource(getResources(), R.mipmap.ic_launcher));

    Intent callIntent = new Intent(Intent.ACTION_DIAL);
    Uri data = Uri.parse("tel:13855271093");
    callIntent.setData(data);
    PendingIntent callPendingIntent = PendingIntent.getActivity(this, 0, callIntent,
PendingIntent.FLAG_UPDATE_CURRENT);
    builder.addAction(new NotificationCompat.Action(R.drawable.ic_launcher,"拨打",callPendingIntent));

    if (Build.VERSION.SDK_INT >= 16) {
        notification = builder.build();
    } else {
        notification = builder.getNotification();
    }

    NotificationManager notificationManager =
            (NotificationManager) getSystemService(NOTIFICATION_SERVICE);
    notificationManager.notify(0, notification);
}
```

这里为了简单起见，并不直接拨号，而是跳转到拨号应用，由用户手动拨号。运行程序后，效果如图 11-11 所示。

点击下面一行任意处，就能够执行拨号任务了，如图 11-12 所示。

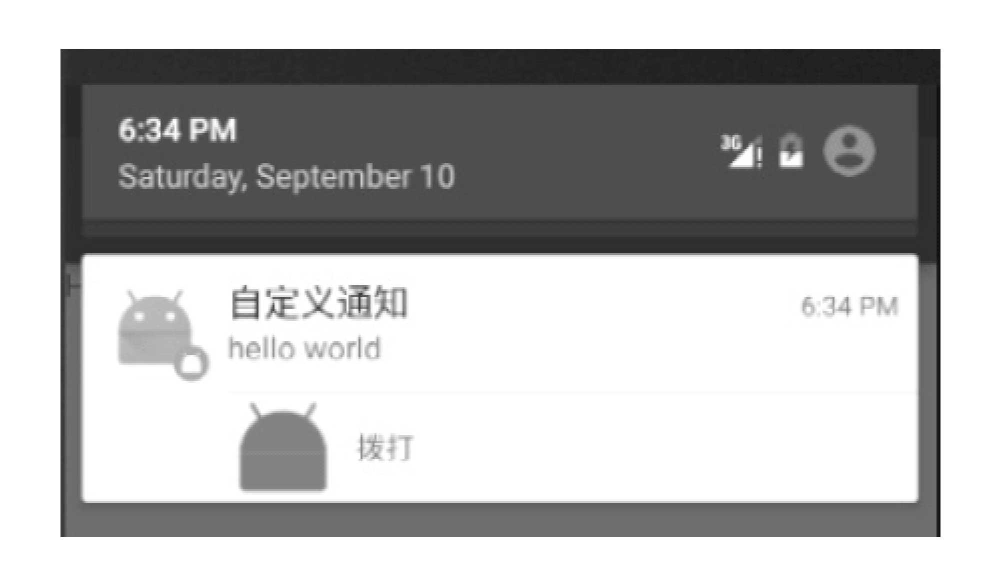


图 11-11　具有可点击按钮的通知

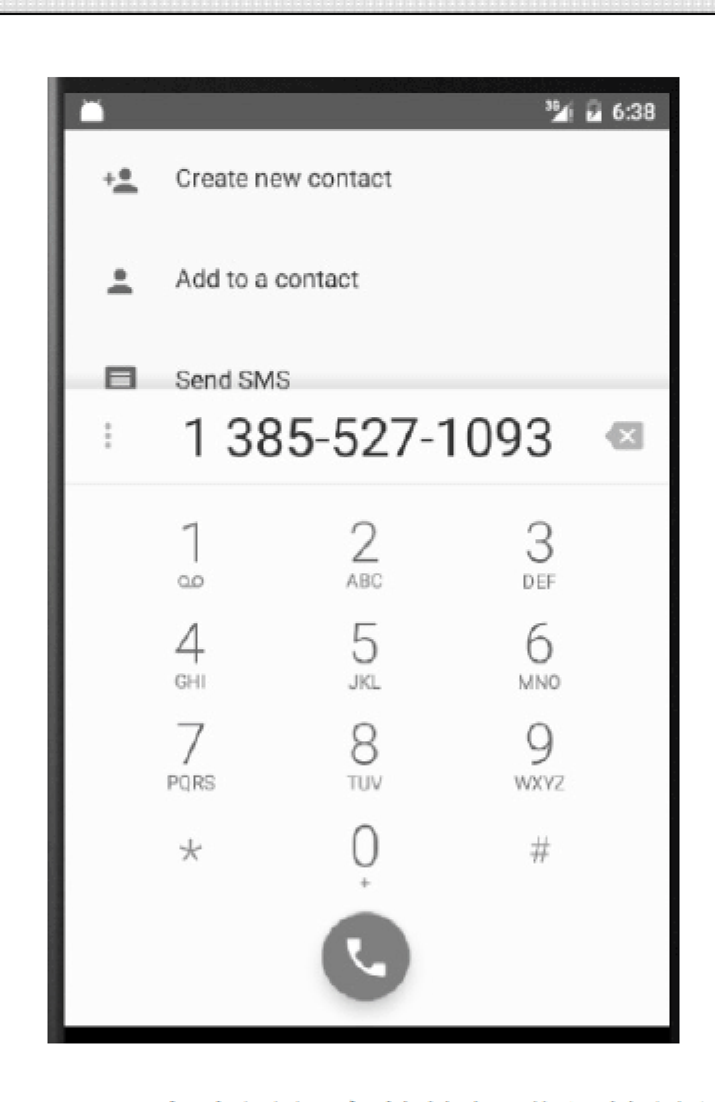


图 11-12　点击通知中的按钮进入拨号界面

2. 带进度条的通知

带进度条的通知经常被用于下载文件时。设置一个带进度条的通知只需要调用 builder 的 setProgress(int max, int progress, boolean indeterminate)即可。下面用一个实例来演示如何使用一个带进度条的通知。修改 MainActivity 如下：

```
public class MainActivity extends AppCompatActivity {

    private NotificationCompat.Builder builder;
    private Notification notification;
    private NotificationManager notificationManager;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        normalNotification("自定义通知", "hello world");
    }

    private void normalNotification(String title, String content) {
        Intent intent = new Intent(this, SecondActivity.class);
        PendingIntent pendingIntent = PendingIntent.getActivity(this, 0, intent,
                                            PendingIntent.FLAG_UPDATE_CURRENT);
        builder = new NotificationCompat.Builder(this);
        builder.setContentIntent(pendingIntent);

        builder.setAutoCancel(true);
        builder.setContentTitle(title);
```

```
        builder.setContentText(content);
        builder.setSmallIcon(R.mipmap.ic_launcher);
        builder.setLargeIcon(BitmapFactory.decodeResource(getResources(), R.mipmap.ic_launcher));

        setDown();
    }

    private void setDown() {
        new Thread(new Runnable() {
            @Override
            public void run() {
                for (int i = 0; i <= 100; i += 5) {
                    builder.setProgress(100, i, false);
                    openNotification();
                    try {
                        Thread.sleep(500);
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                }
                //下载完成
                builder.setContentText("下载完成").setProgress(0, 0, false);
                openNotification();
                try {
                    Thread.sleep(500);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
                //下载完成后，等待 5 秒，关闭通知
                notificationManager.cancel(0);
            }
        }).start();
    }

    private void openNotification() {
        if (Build.VERSION.SDK_INT >= 16) {
            notification = builder.build();
        } else {
            notification = builder.getNotification();
        }

        notificationManager =
                (NotificationManager) getSystemService(NOTIFICATION_SERVICE);
```

```
            notificationManager.notify(0, notification);
        }
    }
```

这里使用多线程根据下载的状况不断去更新通知的内容，并当下载完成时选择关闭通知。运行程序，效果如图 11-13 所示。

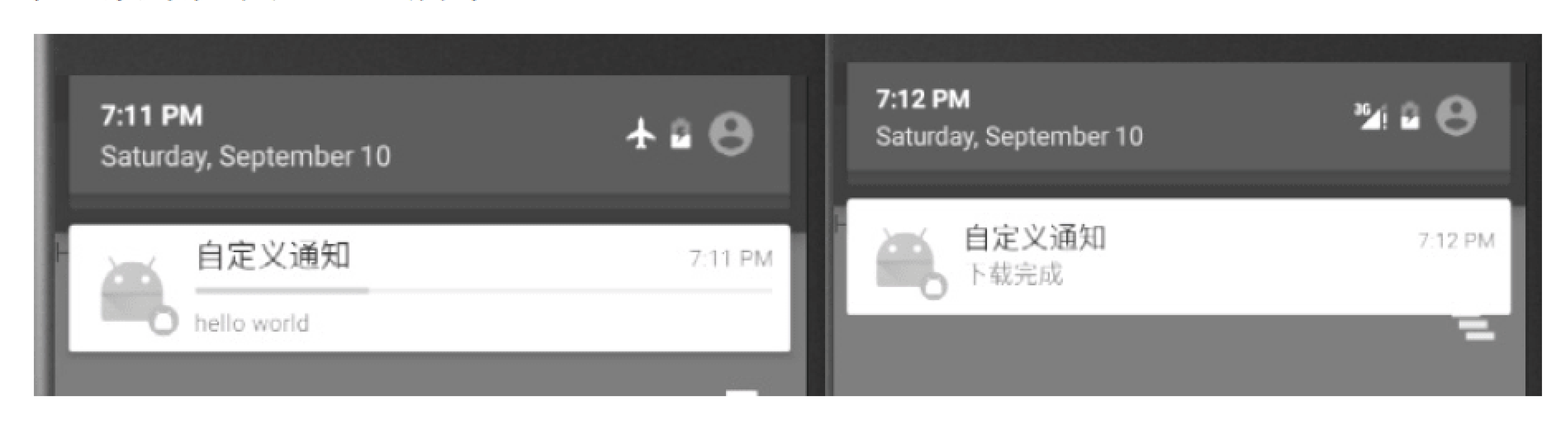


图 11-13　使用通知更新下载状态

当然有的读者可能希望能够展示下载的百分比，这也可以通过多线程来实时更新通知的内容来实现，有兴趣的读者可以进行尝试。

11.3　动　画

动画效果一直是人机交互中非常重要的部分，与死板、突兀的显示效果不同，动画效果的加入，让交互变得更加友好，特别是在提示、引导类的场景中，合理地使用动画能让用户获得更加愉悦的使用体验。在 Android 中，3.0 版本以前支持两种动画模式，补间动画（Tween Animation）和帧动画（Frame Animation），在 3.0 版本中又引入了一个新的动画系统——属性动画（Property Animation）。本节的重点就是讲解这 3 种动画。

11.3.1　帧动画

帧动画，顾名思义就是这个动画的效果是由一帧帧的图片组合出来的。通过指定图片展示的顺序，达到动画的展示效果。一般手机的开机动画、应用的等待动画等都是帧动画，因为只需要几张图片轮播，极其节省资源，如果真的设计成动画，就会很耗费资源。

帧动画可以加载 Drawable 资源实现帧动画。AnimationDrawable 是实现帧动画的基本类，使用此类在代码中控制也可以实现帧动画。官方推荐用 XML 文件的方法实现帧动画，不推荐在代码中实现。要想创建一个帧动画，需要在工程中 res/drawable/目录下新建一个 XML 文件，XML 文件的指令（属性）为动画播放的顺序和时间间隔。在此 XML 文件中<animation-list>元素为根节点，<item>节点定义了每一帧，表示一个 drawable 资源的帧和帧间隔。此 XML 文件必须写在 res 资源文件目录的 anim 文件夹下。

【帧动画实例】

下面通过一个实例来演示如何使用帧动画。创建一个新的工程，在 res/drawable/目录下新建一个 XML 文件 frame_animation.xml，代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<animation-list xmlns:android="http://schemas.android.com/apk/res/android"
    android:oneshot="false">
    <item
        android:drawable="@drawable/horse1"
        android:duration="1000" />
    <item
        android:drawable="@drawable/horse2"
        android:duration="1000" />
    <item
        android:drawable="@drawable/horse3"
        android:duration="1000" />
    <item
        android:drawable="@drawable/horse4"
        android:duration="1000" />
</animation-list>
```

这里面 android:oneshot="false"指的是动画会一直循环运行，不会自动停止，如果值为 true 时，就会在运行一次之后自动停止动画。在 item 标签中，drawable 的属性自然是指的帧动画对应的图片资源，而 duration 属性则是指的该图片显示的时间，单位为毫秒。

接下来修改布局文件 activity_main.xml。为了展示动画图片在布局文件中加入了一个 ImageView，同时为了控制动画的开启与停止，增加了一个 Button，代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context="com.buaa.animation.MainActivity">

    <ImageView
        android:id="@+id/animation_img"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />

    <Button
        android:id="@+id/ani_button"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="开启动画"/>
</LinearLayout>
```

最后修改 MainActivity 文件。在 MainActivity 中获取布局文件中的控件，并给 ImageView 设置动画，同时点击按钮时开始或停止动画，代码如下：

```
public class MainActivity extends AppCompatActivity {

    private ImageView imageView;
    private Button button;
    private boolean flag = true;
    private AnimationDrawable animationDrawable;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        imageView = (ImageView) findViewById(R.id.animation_img);
        imageView.setImageResource(R.drawable.frame_animation);
        animationDrawable = (AnimationDrawable) imageView.getDrawable();
        button = (Button) findViewById(R.id.ani_button);
        button.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                if (flag) {
                    button.setText("停止动画");
                    animationDrawable.start();
                    flag = false;
                } else {
                    button.setText("开启动画");
                    animationDrawable.stop();
                    flag = true;
                }
            }
        });
    }
}
```

运行程序，就会发现开始时只显示一张静态图片，点击按钮之后，动画效果就会出现，再次点击按钮则停止动画。由于是动画效果，不便展示，因此这里不再展示效果图。

11.3.2 补间动画

组件由原始状态向终极状态转变时，为了让过渡更自然而自动生成的动画叫作补间动画。补间动画与逐帧动画在本质上是不同的，逐帧动画通过连续播放图片来模拟动画的效果，而补间动画则是通过在两个关键帧之间补充渐变的动画效果来实现的。补间动画的优点是可以节省

空间，当然缺点也是明显的。补间动画虽然可能改变了控件的位置，但是控件的实际属性值未变，比如动画移动一个按钮位置，但按钮点击的实际位置仍未改变。所以一般来说，补间动画适用于一些最终位置不发生变化的动画效果。目前 Android 应用框架支持的补间动画效果有以下 5 种，具体实现在 android.view.animation 类库中：

- AlphaAnimation：透明度（alpha）渐变效果，对应<alpha/>标签。
- TranslateAnimation：位移渐变，需要指定移动点的开始和结束坐标，对应<translate/>标签。
- ScaleAnimation：缩放渐变，可以指定缩放的参考点，对应<scale/>标签。
- RotateAnimation：旋转渐变，可以指定旋转的参考点，对应<rotate/>标签。
- AnimationSet：组合渐变，支持组合多种渐变效果，对应<set/>标签。

补间动画的效果同样可以使用 XML 语言来定义，这些动画模板文件通常会被放在 Android 项目的 res/anim/目录下。下面我们先在 res/anim 中新建 5 个分别对应上述 5 种效果的动画模板文件。

透明度效果

```
<?xml version="1.0" encoding="utf-8"?>
<alpha xmlns:android="http://schemas.android.com/apk/res/android"
   android:duration="2000"
   android:fromAlpha="1.0"
   android:toAlpha="0.1    " />
```

其中，fromAlpha 属性表示起始透明度，属性值为 0 到 1 之间的数，1 表示完全不透明，0 表示完全透明。toAlpha 属性表示动画结束时的透明度，属性值与 fromAlpha 相同。Duration 属性表示动画持续时长，此处的 2000 表示 2000 毫秒，其他类型的动画中此属性表示意义相同。

位移效果

```
<?xml version="1.0" encoding="utf-8"?>
<translate xmlns:android="http://schemas.android.com/apk/res/android"
      android:duration="2000"
      android:fromXDelta="0"
      android:fromYDelta="0"
      android:toXDelta="50"
      android:toYDelta="-50" />
```

fromXDelta 属性指动画起始位置的横坐标，toXDelta 代表动画结束位置的横坐标；fromYDelta 意味着动画起始位置的纵坐标，toYDelta 指动画结束位置的纵坐标。

缩放效果

```
<?xml version="1.0" encoding="utf-8"?>
<scale xmlns:android="http://schemas.android.com/apk/res/android"
   android:duration="2000"
   android:fromXScale="0.2    "
```

```
    android:fromYScale="0.2"
    android:pivotX="50%"
    android:pivotY="50%"
    android:toXScale="1.5"
    android:toYScale="1.5" />
```

fromXScale 属性表示沿着 *x* 轴缩放的起始比例，toXScale 表示沿着 *x* 轴缩放的结束比例。fromYScale 属性表示沿着 *y* 轴缩放的起始比例，toYScale 属性表示沿着 *y* 轴缩放的结束比例。pivotX 和 pivotY 分别代表着图片缩放的中心点位置。

旋转效果

```
<?xml version="1.0" encoding="utf-8"?>
<rotate xmlns:android="http://schemas.android.com/apk/res/android"
    android:duration="1000"
    android:fromDegrees="0"
    android:repeatCount="1"
    android:repeatMode="reverse"
    android:toDegrees="360" />
```

其中，fromDegrees 属性表示旋转的起始角度，toDegrees 属性表示旋转的结束角度。repeatCount 属性用于设置旋转的次数，默认值是 0，代表旋转 1 次。如果值 repeatCount=4 则表示需要旋转 5 次，值为-1 或者 infinite 时，表示补间动画将永不停止。repeatMode 属性用于设置重复的模式，默认是 restart，还可以设成 reverse，表示偶数次显示动画时会做与动画文件定义方向相反的动作。另外，repeatMode 属性只有当 repeatCount 的值大于 0 或者为 infinite 时才有效。

组合动画

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android"
  android:interpolator="@android:anim/decelerate_interpolator"
  android:shareInterpolator="true">

  <scale
    android:duration="2000"
    android:fromXScale="0.2"
    android:fromYScale="0.2"
    android:pivotX="50%"
    android:pivotY="50%"
    android:toXScale="1.5"
    android:toYScale="1.5" />

  <rotate
    android:duration="1000"
```

```
        android:fromDegrees="0"
        android:repeatCount="1"
        android:repeatMode="reverse"
        android:toDegrees="360" />

    <translate
        android:duration="2000"
        android:fromXDelta="0"
        android:fromYDelta="0"
        android:toXDelta="200"
        android:toYDelta="200" />

    <alpha
        android:duration="2000"
        android:fromAlpha="1.0"
        android:toAlpha="0.1    " />
</set>
```

这种组合效果是对其他 4 种效果的组合，开发时可根据需要进行随意组合。

下面新建一个 Activity 类 TweenActivity 来使用这些补间动画。修改 activity_tween.xm：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:gravity="center"
    android:orientation="vertical"
    tools:context="com.buaa.animation.TweenActivity">

    <ImageView
        android:id="@+id/tween_animation"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@drawable/ic_launcher" />

    <Button
        android:id="@+id/alpha"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="透明度动画" />

    <Button
        android:id="@+id/translate"
```

```
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="位移动画" />

        <Button
            android:id="@+id/scale"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="缩放动画" />

        <Button
            android:id="@+id/rotate"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="旋转动画" />

        <Button
            android:id="@+id/set"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="组合动画" />
</LinearLayout>
```

这里使用了 5 个按钮以及一个 ImageView，当点击不同按钮时使 ImageView 按照不同的效果动起来。接下来在 Activity 中获取这些按钮控件，并触发相应的动画效果。代码如下：

```
public class TweenActivity extends AppCompatActivity implements View.OnClickListener {

    private Button alphaButton;
    private Button translateButton;
    private Button scaleButton;
    private Button rotateButton;
    private Button setButton;
    private ImageView imageView;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_tween);
        initView();
    }

    private void initView() {
        imageView = (ImageView) findViewById(R.id.tween_animation);
```

```
        alphaButton = (Button) findViewById(R.id.alpha);
        translateButton = (Button) findViewById(R.id.translate);
        scaleButton = (Button) findViewById(R.id.scale);
        rotateButton = (Button) findViewById(R.id.rotate);
        setButton = (Button) findViewById(R.id.set);

        alphaButton.setOnClickListener(this);
        translateButton.setOnClickListener(this);
        scaleButton.setOnClickListener(this);
        rotateButton.setOnClickListener(this);
        setButton.setOnClickListener(this);
    }

    private void startAnimation(int anim) {
        Animation animation = AnimationUtils.loadAnimation(this, anim);
        //设置是否保持在最终状态，true 为是
        animation.setFillAfter(true);
        imageView.startAnimation(animation);
    }

    @Override
    public void onClick(View v) {
        switch (v.getId()) {
            case R.id.alpha:
                startAnimation(R.anim.alpha);
                break;
            case R.id.translate:
                startAnimation(R.anim.translate);
                break;
            case R.id.scale:
                startAnimation(R.anim.scale);
                break;
            case R.id.rotate:
                startAnimation(R.anim.rotate);
                break;
            case R.id.set:
                startAnimation(R.anim.set);
                break;
        }
    }
}
```

代码中通过 AnimationUtils 的静态方法 loadAnimation()获取对应动画模板的 Animation 类，然后用 ImageView 的 startAnimation()开启相应的动画效果。运行程序，通过点击各按钮，就可以触发对应的动画效果。这里不做演示。

在实际项目中，我们经常使用补间动画，原因是补间动画使用起来比较方便，功能也比逐帧动画强很多，而且还可以很方便地进行动画叠加，实现更加复杂的效果。

11.3.3 属性动画

通过上面的学习，读者可能会觉得补间动画功能很强大，实现的功能比较完善。实事求是地说，补间动画已经相当强大了，但是它依旧存在一些缺陷。具体来说，补间动画的缺陷有如下 3 个。

第一，补间动画是只能够作用在 View 上的。也就是说，使用补间动画可以对一个 Button、TextView、甚至是 LinearLayout 或者其他任何继承自 View 的组件进行动画操作，但是不能对一个非 View 的对象进行动画操作。比如，在一个自定义的 View 中有多个组成部分，补间动画就无法给其中的某个部分设置动画效果。

第二，补间动画只能够实现移动、缩放、旋转和淡入淡出这 4 种动画操作，无法动态地改变 View 的背景色。

第三，补间动画最致命的缺陷是它只改变了 View 的显示效果，而不会真正改变 View 的属性。比如说，现在屏幕的左上角有一个按钮，当通过补间动画将它移动到屏幕的右下角时，尝试点击一下这个按钮，会发现点击事件是绝对不会触发的。这是因为实际上这个按钮还是停留在屏幕的左上角，只不过补间动画将这个按钮绘制到了屏幕的右下角而已。

为了解决上述问题，Android 在 3.0 版本当中引入了属性动画。属性动画机制不再针对 View 来设计，也不限定于只能实现移动、缩放、旋转和淡入淡出这几种动画操作，同时也不再只是一种视觉上的动画效果了。它实际上是一种不断地对值进行操作的机制，并将值赋到指定对象的指定属性上，可以是任意对象的任意属性。所以我们仍然可以将一个 View 进行移动或者缩放，但同时也可以对自定义 View 中的某个部分进行动画操作。开发者只需要告诉系统动画的运行时长，需要执行哪种类型的动画，以及动画的初始值和结束值，剩下的工作系统会自动完成。

【补间动画实例】

通过上述介绍，读者应该已经对属性动画有了一个最基本的认识，下面我们就通过实例来学习如何使用属性动画。本例将使用属性动画来实现补间动画所实现的几种效果。创建一个新的 Activity 类 PropertyActivity 类。PropertyActivity 的布局文件与 TweenActivity 类的布局文件一样，Java 代码部分只有 OnClick()有所区别，PropertyActivity 类中的 onClick()方法如下：

```
@Override
public void onClick(View v) {
    switch (v.getId()) {
        case R.id.alpha:
            ObjectAnimator alpha = ObjectAnimator.
```

```
                    ofFloat(imageView, "alpha", 1, 0, 1);
            alpha.setDuration(2000);
            alpha.start();
            break;
        case R.id.translate:
            //如果使用 translationX 或者 translationY 将沿某一个方向位移
            ObjectAnimator translate = ObjectAnimator.
                    ofFloat(imageView, "translationX", 0, -500, 0);
            translate.setDuration(2000);
            translate.start();
            break;
        case R.id.scale:
            //如果使用 scaleX 或者 scaleY 将沿某一个方向缩放
            ObjectAnimator scale = ObjectAnimator.
                    ofFloat(imageView, "scaleX", 1, 0);
            scale.setDuration(2000);
            scale.start();
            break;
        case R.id.rotate:
            ObjectAnimator rotate = ObjectAnimator.
                    ofFloat(imageView, "rotation", 0, 360);
            rotate.setDuration(2000);
            rotate.start();
            break;
        case R.id.set:
            //如果使用 rotationX 或者 rotationY 将会向某一个方向旋转
            ObjectAnimator moveIn = ObjectAnimator.
                    ofFloat(imageView, "translationX", -500, 0);
            ObjectAnimator rotation = ObjectAnimator.
                    ofFloat(imageView, "rotation", 0, 360);
            ObjectAnimator fadeInOut = ObjectAnimator.
                    ofFloat(imageView, "alpha", 1, 0, 1);
            AnimatorSet animSet = new AnimatorSet();
            animSet.play(rotation).with(fadeInOut).after(moveIn);
            animSet.setDuration(2000);
            animSet.start();
    }
}
```

读者会发现每种动画的实现其实都很简单，只需要使用 ObjectAnimator 的静态方法 ofFloat()方法获得 ObjectAnimator 类的对象，然后设置时间参数后调用 start()方法开启动画即可。在 ofFloat()方法的几个参数中，第一个是想要参加动画的对象，第二个参数为想要执行的动画类型，剩下的参数为开始位置、结束位置、缩放比例等内容。对于组合动画来说，只需使

用 animSet.play(rotation).with(fadeInOut).after(moveIn)即可。

运行程序，会发现通过属性动画确实实现了补间动画的效果，而且避免了补间动画的缺陷。另外，属性动画除了能够实现基本的动画外，还可以对动画运行的过程进行监听，方法如下（这里并没有实现具体的处理方式）：

```
alpha.addListener(new Animator.AnimatorListener() {
    @Override
    public void onAnimationStart(Animator animation) {
    }
    @Override
    public void onAnimationEnd(Animator animation) {
    }
    @Override
    public void onAnimationCancel(Animator animation) {
    }
    @Override
    public void onAnimationRepeat(Animator animation) {
    }
});
```

属性动画的内容很多，本文介绍的属性动画内容虽然还不是非常全面，但是已经可以基本满足开发需要，如果有读者想要了解更多内容，可以访问官方文档进行学习。

11.4　相机与相册

很多应用程序都可能会使用到调用相机拍照的功能，比如说微博用户突然觉得眼前风景很美，想要发一条微博，这时在应用中打开相机拍张照是最简单快捷的。

11.4.1　相机的使用

在 Android 中调用系统相机很简单，只需使用隐式意图的方式来打开系统相机应用即可。根据 Action 的不同，调用相机的隐式意图可以分两种，一种是将 Intent 构造方法的参数设为"android.media.action.STILL_IMAGE_CAMERA"，即 Intent intent = new Intent("android.media.action.STILL_IMAGE_CAMERA")；另一种是将 Intent 构造方法的参数设为"android.media.action.IMAGE_CAPTURE"，即 Intent intent = new Intent("android.media.action.IMAGE_CAPTURE"。前者调用系统应用后将一直留在相机界面，而且没有数据返回。后者则不一样，它将会返回拍照之后产生的相关数据，比如照片等，而拍摄之后也会直接返回当前应用中。

在实际应用中，一般都是拍摄完一张照片之后就需要返回到当前应用，所以后一种方式使用的较多，本节讲解的相机也是后一种。当然也不能排除有的场景下是需要连续拍照的，这时就需要使用前一种方式来调用相机了。除此之外，Android 中还可以使用自定义相机，不过这种方式相对较为复杂，而系统相机也能够满足大部分的需求，同时由于手机厂商对相机定制

化的修改，自定义相机时会出现诸多的不兼容问题，因此并不建议自定义相机。所以本节将主要讲解如何使用系统相机。

下面通过实例来进行学习。首先创建一个项目 camera，修改布局文件 activity_main.xml：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:gravity="center_horizontal"
    android:orientation="vertical"
    tools:context="com.buaa.camera.MainActivity">

    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="30dp"
        android:text="点击拍照" />

    <ImageView
        android:id="@+id/image"
        android:layout_width="300dp"
        android:layout_height="300dp"
        android:layout_marginTop="40dp" />
</LinearLayout>
```

这里添加了一个 Button 按钮，用于打开相机，同时加入了一个 ImageView，用于展示拍摄的内容。下面通过修改 MainActivity 来进行调用系统相机并处理数据的操作，代码如下：

```
public class MainActivity extends AppCompatActivity implements View.OnClickListener {

    private ImageView imageView;
    private Button button;
    private final int IMAGE_CAMERA = 123;
    private final int PERMISSION_CODE = 122;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        intiView();
    }

    private void intiView() {
```

```
        button = (Button) findViewById(R.id.button);
        imageView = (ImageView) findViewById(R.id.image);
        button.setOnClickListener(this);
    }

    @Override
    public void onClick(View view) {
        getCameraPermission();
    }

    private void openCamera() {
        Intent intent = new Intent("android.media.action.IMAGE_CAPTURE");
        startActivityForResult(intent, IMAGE_CAMERA);
    }

    @Override
    protected void onActivityResult(int requestCode, int resultCode, Intent data) {
        super.onActivityResult(requestCode, resultCode, data);
        if (requestCode == IMAGE_CAMERA && resultCode == RESULT_OK) {
            Bundle bundle = data.getExtras();
            Bitmap bitmap = (Bitmap) bundle.get("data");
            imageView.setImageBitmap(bitmap);
        }
    }

    public void getCameraPermission() {
        if (Build.VERSION.SDK_INT >= 23) {
            int checkPermission = ContextCompat.
                    checkSelfPermission(this, Manifest.permission.CAMERA);
            if (checkPermission != PackageManager.PERMISSION_GRANTED) {
                ActivityCompat.requestPermissions(this,
                        new String[]{Manifest.permission.CAMERA},
                        PERMISSION_CODE);
                return;
            } else {
                openCamera();
            }
        } else {
            openCamera();
        }
    }

    @Override
    public void onRequestPermissionsResult(int requestCode, String[] permissions, int[] grantResults) {
```

```
            switch (requestCode) {
                case PERMISSION_CODE:
                    if (grantResults[0] == PackageManager.PERMISSION_GRANTED) {
                        Toast.makeText(this, "获取权限成功", Toast.LENGTH_SHORT)
                                .show();
                        openCamera();
                    } else {
                        Toast.makeText(this, "获取权限失败", Toast.LENGTH_SHORT)
                                .show();
                    }
                    break;
                default:
                    super.onRequestPermissionsResult(requestCode, permissions, grantResults);
            }
        }
    }
```

这里的处理很简单，只是通过 startActivityForResult() 调用了系统相机，然后在 onActivityResult()方法中获取拍照时返回的数据，再显示到 ImageView 上。同时，调用相机是危险权限，所以这里需要动态地获取权限，当然还应该在 AndroidManifest.xml 中声明此权限：

```
<uses-permission android:name="android.permission.CAMERA"/>
```

这时运行程序，点击按钮时会先提示是否允许拍照，点击允许之后就会进入相机界面，此时就可以进行拍照了。拍完之后效果如图 11-14 所示。

点击对号选择此张照片，就会从相机界面返回之前的界面，同时照片会显示到 ImageView 上，如图 11-15 所示。

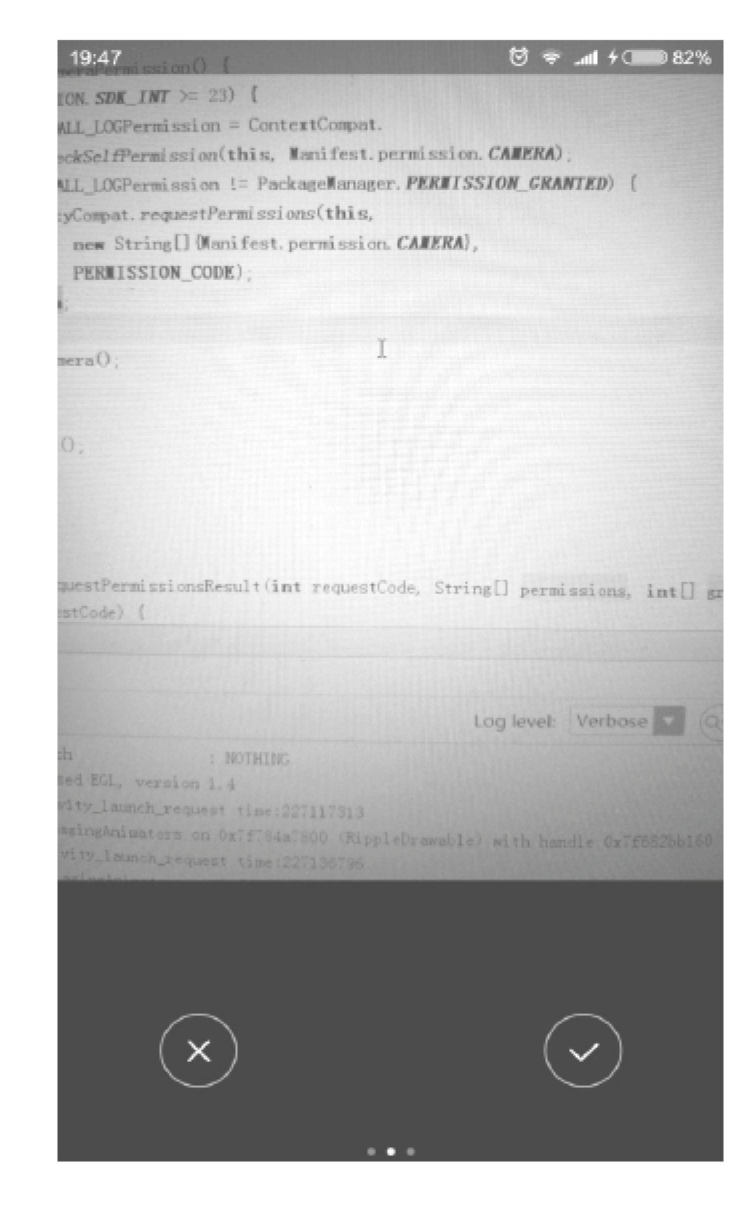


图 11-14　使用相机拍完照片时的效果

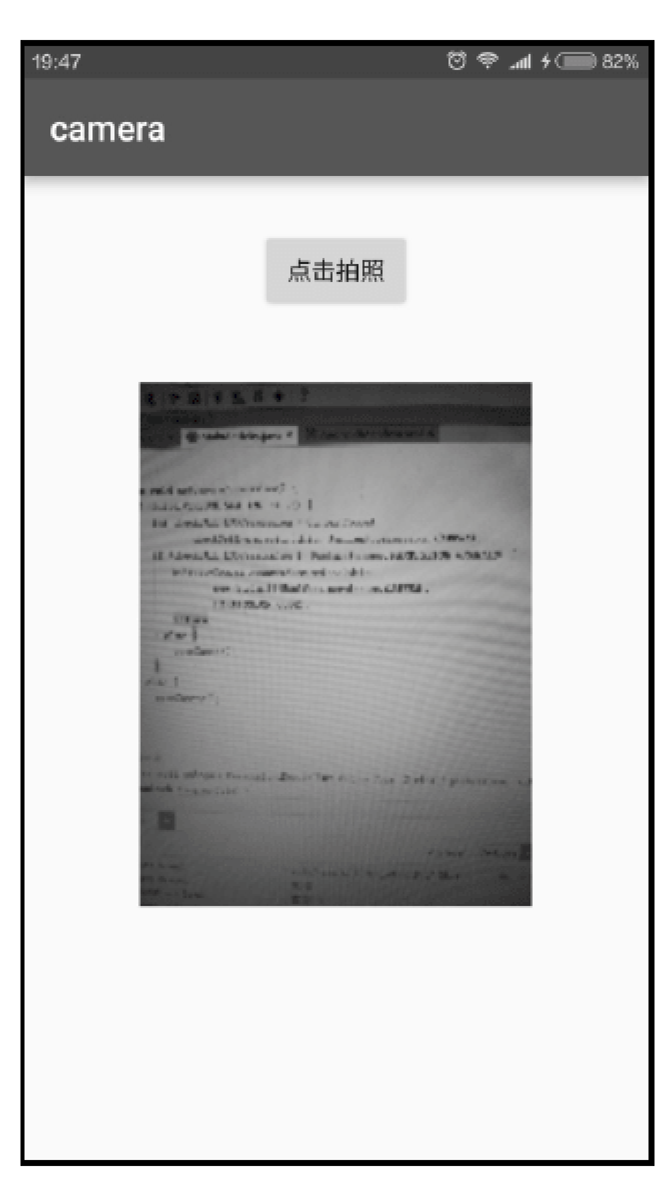


图 11-15　被选择的照片显示到 ImageView 中

11.4.2 相册的使用

在很多场景中需要实现上传照片的功能，使用系统相机及时拍一张是很好的选择，但是如果此时系统相册中如果已经存在此照片，直接上传岂不是更好吗？这时就需要使用到系统相册了。

使用相册的方法有很多种，比如下面这种方式：

```
Intent intent = new Intent(Intent.ACTION_GET_CONTENT);
intent.setType("image/*");
startActivityForResult(intent, IMAGE_ALBUM);
```

这种方式在一些书籍中被作为范例使用，而在 Android 4.4 版本之前也确实被经常使用，但是之后的版本中以这种方式打开的文件，当选中图片之后，在 onActivityResult()方法中会出现接收不到图片甚至崩溃的状况。因此，本书中并不推荐使用这种方式。除此之外，还有一种方式比较常用：

```
Intent intent = new Intent(Intent.ACTION_PICK);
intent.setType("image/*");
startActivityForResult(intent, IMAGE_ALBUM);
```

这种方式的问题和上一种比较相似，也会出现各种版本兼容问题。因此在 Android 4.4 版本之后，官方推荐使用下面这种方式来获取图片，但是由于市场还会有 4.4 以下版本的手机存在，所以还应对不同系统进行不同处理：

```
Intent intent = new Intent();
    intent.addCategory(Intent.CATEGORY_OPENABLE);
    intent.setType("image/*");
    //根据版本号不同使用不同的 Action
    if (Build.VERSION.SDK_INT < 19) {
        intent.setAction(Intent.ACTION_GET_CONTENT);
    } else {
        intent.setAction(Intent.ACTION_OPEN_DOCUMENT);
    }
    startActivityForResult(intent, IMAGE_ALBUM);
```

这种方式打开的其实是系统文件夹，用户可以在文件夹中选择图片。选择图片完成后在 onActivityResult()方法处理即可。下面通过实例来说明它的使用方法。由于其他部分代码与上一部分的实例基本一致，因此这里只列出核心部分代码：

```
@Override
public void onClick(View view) {
    openImageFile();
}

private void openImageFile() {
```

```
        Intent intent = new Intent();
        intent.addCategory(Intent.CATEGORY_OPENABLE);
        intent.setType("image/*");
        //根据版本号使用不同的 Action
        if (Build.VERSION.SDK_INT < 19) {
            intent.setAction(Intent.ACTION_GET_CONTENT);
        } else {
            intent.setAction(Intent.ACTION_OPEN_DOCUMENT);
        }
        startActivityForResult(intent, IMAGE_ALBUM);
    }

    @Override
    protected void onActivityResult(int requestCode, int resultCode, Intent data) {
        super.onActivityResult(requestCode, resultCode, data);
        if (resultCode == RESULT_OK) {
            switch (requestCode) {
                case IMAGE_ALBUM:
                    Bitmap albumBitmap = data.getParcelableExtra("data");
                    imageView.setImageBitmap(albumBitmap);
                    break;
            }
        }
    }
```

这里的代码并不复杂，且与上一部分相似，这里不再讲解。运行程序，点击按钮之后会进入文件夹，此时进入任意一个包含图片的文件夹均可，然后选择该图片，如图 11-16 所示。完成之后会回到当前应用，图片显示到 UI 界面，如图 11-17 所示。

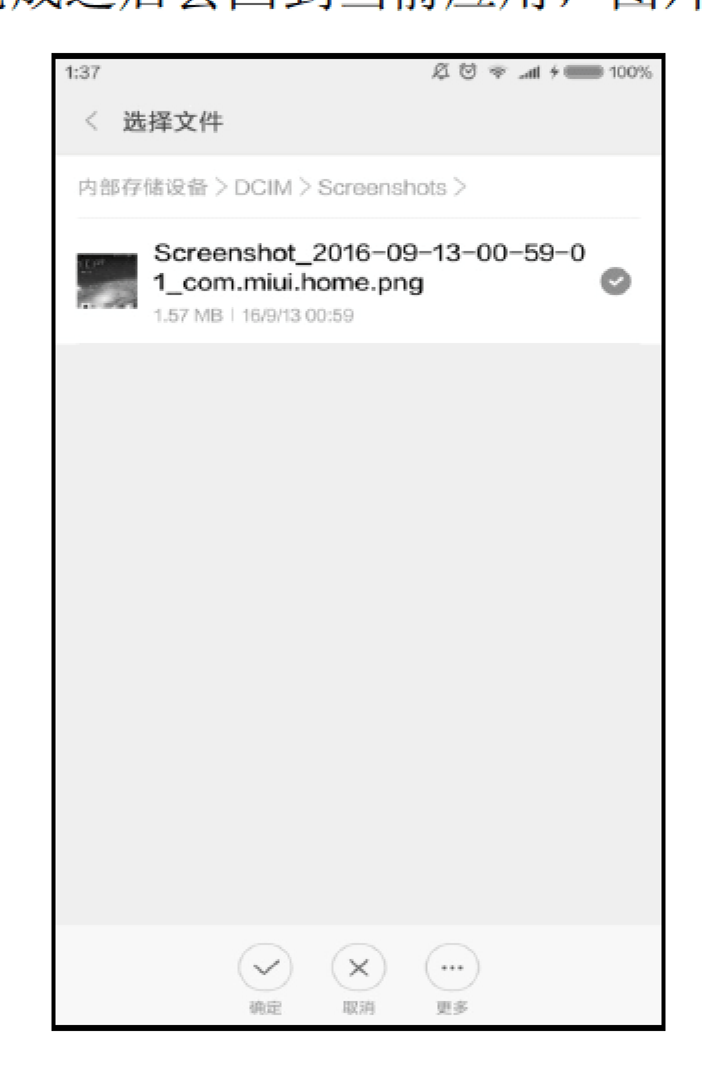


图 11-16　从本地选择图片



图 11-17　被选择的照片显示到 UI 界面中

11.4.3 图片的裁剪

不管是拍照获取照片还是直接从相册获取照片，有时都不能直接使用，还需对此做一些裁剪。对图片进行裁剪也很简单，只需要通过隐式意图调用系统的裁剪器即可，方法如下：

```
private void cutImage(Bitmap bitmap) {
    Intent intent = new Intent();
    intent.setAction("com.android.camera.action.CROP");
    intent.setType("image/*");
    intent.putExtra("data", bitmap);
    intent.putExtra("crop", "true");
    intent.putExtra("aspectX", 1);// 裁剪框比例
    intent.putExtra("aspectY", 1);
    intent.putExtra("outputX", 150);// 输出图片大小
    intent.putExtra("outputY", 150);
    intent.putExtra("return-data", true);
    startActivityForResult(intent, CUT_PHOTO);
}
```

"com.android.camera.action.CROP"就是裁剪器所对应的 action，通过它就可以调用系统的裁剪功能。这里需要传递一些参数，其中最主要的是将数据 bitmap 通过 intent.putExtra("data", bitmap)传给裁剪器。这里给出一个完整的拍照、裁剪，从本地获取图片、裁剪的实例。在 activity_main.xml 中使用两个按钮和一个 ImageView，分别用于触发拍照、从本地获取图片和显示图片，代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:gravity="center_horizontal"
    android:orientation="vertical"
    tools:context="com.buaa.camera.MainActivity">

    <Button
        android:id="@+id/album"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="30dp"
        android:text="获取拍照" />

    <Button
        android:id="@+id/camera"
        android:layout_width="wrap_content"
```

```
        android:layout_height="wrap_content"
        android:text="拍摄拍照" />

    <ImageView
        android:id="@+id/image"
        android:layout_width="300dp"
        android:layout_height="300dp"
        android:layout_marginTop="40dp" />
</LinearLayout>
```

在 Activity 中获取控件，设置按钮的点击事件分别为拍照和获取本地图片，并在 onActivityResult()中处理两者都成功后的结果。这里并不直接显示到 ImageView 中，而是先进行裁剪处理再显示，代码如下：

```
package com.buaa.camera;

import android.Manifest;
import android.content.Intent;
import android.content.pm.PackageManager;
import android.graphics.Bitmap;
import android.os.Build;
import android.support.v4.app.ActivityCompat;
import android.support.v4.content.ContextCompat;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.ImageView;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity implements View.OnClickListener {

    private ImageView imageView;
    private Button cameraButtbon;
    private Button albumButton;
    private final int IMAGE_CAMERA = 123;
    private final int CUT_PHOTO = 124;
    private final int PERMISSION_CODE = 122;
    private final int IMAGE_ALBUM = 125;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
```

```
        intiView();
    }

    private void intiView() {
        cameraButtbon = (Button) findViewById(R.id.camera);
        albumButton = (Button) findViewById(R.id.album);
        imageView = (ImageView) findViewById(R.id.image);
        cameraButtbon.setOnClickListener(this);
        albumButton.setOnClickListener(this);
    }

    @Override
    public void onClick(View view) {
        switch (view.getId()) {
            case R.id.album:
                openImageFile();
                break;
            case R.id.camera:
                getPermission();
                break;
        }
    }

    @Override
    protected void onActivityResult(int requestCode, int resultCode, Intent data) {
        super.onActivityResult(requestCode, resultCode, data);
        if (resultCode == RESULT_OK) {
            switch (requestCode) {
                case IMAGE_CAMERA:
                    Bitmap cameraBitmap = data.getParcelableExtra("data");
                    cutImage(cameraBitmap);
                    break;
                case IMAGE_ALBUM:
                    Bitmap albumBitmap = data.getParcelableExtra("data");
                    cutImage(albumBitmap);
                    break;
                case CUT_PHOTO:
                    Bitmap cutBitmap = data.getParcelableExtra("data");
                    imageView.setImageBitmap(cutBitmap);
            }
        }
    }
```

```
    private void cutImage(Bitmap bitmap) {
        Intent intent = new Intent();
        intent.setAction("com.android.camera.action.CROP");
        intent.setType("image/*");
        intent.putExtra("data", bitmap);
        intent.putExtra("crop", "true");
        intent.putExtra("aspectX", 1);// 裁剪框比例
        intent.putExtra("aspectY", 1);
        intent.putExtra("outputX", 150);// 输出图片大小
        intent.putExtra("outputY", 150);
        intent.putExtra("return-data", true);
        startActivityForResult(intent, CUT_PHOTO);
    }

    private void openImageFile() {
        Intent intent = new Intent();
        intent.addCategory(Intent.CATEGORY_OPENABLE);
        intent.setType("image/*");
        //根据版本号使用不同的 Action
        if (Build.VERSION.SDK_INT < 19) {
            intent.setAction(Intent.ACTION_GET_CONTENT);
        } else {
            intent.setAction(Intent.ACTION_OPEN_DOCUMENT);
        }
        startActivityForResult(intent, IMAGE_ALBUM);
    }

    private void openCamera() {
        Intent intent = new Intent("android.media.action.IMAGE_CAPTURE");
        startActivityForResult(intent, IMAGE_CAMERA);
    }

    public void getPermission() {
        if (Build.VERSION.SDK_INT >= 23) {
            int checkPermission = ContextCompat.
                    checkSelfPermission(this, Manifest.permission.CAMERA);
            if (checkPermission != PackageManager.PERMISSION_GRANTED) {
                ActivityCompat.requestPermissions(this,
                        new String[]{Manifest.permission.CAMERA},
                        PERMISSION_CODE);
                return;
            } else {
                openCamera();
```

```
                }
            } else {
                openCamera();
            }
        }

        @Override
        public void onRequestPermissionsResult(int requestCode, String[] permissions,
                                                int[] grantResults) {
            switch (requestCode) {
                case PERMISSION_CODE:
                    if (grantResults[0] == PackageManager.PERMISSION_GRANTED) {
                        Toast.makeText(this, "获取权限成功", Toast.LENGTH_SHORT)
                                .show();
                        openCamera();
                    } else {
                        Toast.makeText(this, "获取权限失败", Toast.LENGTH_SHORT)
                                .show();
                        this.finish();
                    }
                    break;
                default:
                    super.onRequestPermissionsResult(requestCode, permissions, grantResults);
            }
        }
    }
```

读者可能会发现这其实是将前两部分的内容进行综合之后加入了裁剪的功能。运行程序，点击拍照或者从本地获取图片的效果与前两部分是相同的，只是在确定选择该图片之后会先跳转到裁剪界面，如图 11-18 所示。

此时点击“应用”将会自动回到当前应用，并在 ImageView 中显示经过裁剪的照片。

本节内容中讲解的相机和相册的相关知识并未能涵盖所有的知识点，虽然可以应对基本的需求，但不足以解决所有相关问题，因此读者在学习本节时一定要通过阅读文档来扩展相关知识。



图 11-18　裁剪图片

11.5 媒体播放器的开发

今天，智能手机早已取代了 MP3 播放器、MP4 播放器成为播放音频和视频的最佳选择。Android 系统在播放音频和视频方面进行了非常强大的支持，开发者只需要使用系统 API 就能够轻松地编写出一个简易的音频和视频播放器。本节内容将主要讲解如何实现音频和视频的播放。

11.5.1 开发一个音频播放器

在 Android 中播放音频文件一般都是使用 MediaPlayer 类来实现的，它对多种格式的音频文件提供了非常全面的控制方法，从而使得播放音乐的工作变得十分简单。MediaPlayer 类的常用方法如表 11-2 所示。

表11-2 MediaPlayer类的常用方法

方法名称	方法描述
setDataSource()	设置要播放的音频文件的位置
prepare()	在开始播放之前调用这个方法完成准备工作
start()	开始或继续播放音频
pause()	暂停播放音乐
seekTo()	从指定的位置开始播放音频
release()	释放掉与 MediaPlayer 对象相关的资源
stop()	停止播放音频，调用这个方法后的 MediaPlayer 对象无法再播放音频
isPlaying(	判断当前 MediaPlayer 是否正在播放音频
getDuration()	获取载入的音频文件的时长
reset()	将 MediaPlayer 对象重置到刚刚创建的状态

下面通过一个实例来学习如何使用 MediaPlayer 播放音乐。创建一个新的工程，修改 activity_main.xml：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:gravity="center"
    android:orientation="vertical"
    tools:context="com.buaa.media.MainActivity">

    <TextView
        android:layout_width="wrap_content"
```

```
        android:layout_height="0dp"
        android:layout_weight="2"
        android:paddingTop="45dp"
        android:text="李子熟了音乐播放器"
        android:textSize="24sp" />

    <TextView
        android:id="@+id/song_name"
        android:layout_width="wrap_content"
        android:layout_height="0dp"
        android:layout_weight="5"
        android:paddingTop="45dp"
        android:textSize="22sp" />

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_gravity="center_vertical"
        android:layout_weight="1"
        android:orientation="horizontal">

        <TextView
            android:id="@+id/played_time"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="00:00"
            android:textSize="16dp" />

        <SeekBar
            android:id="@+id/seek_bar"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_weight="7" />

        <TextView
            android:id="@+id/all_time"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="00:00"
            android:textSize="16dp" />
    </LinearLayout>
```

```
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="2"
        android:orientation="horizontal">

        <Button
            android:id="@+id/play"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="开始" />

        <Button
            android:id="@+id/pause"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="暂停" />

        <Button
            android:id="@+id/stop"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="停止" />
    </LinearLayout>
</LinearLayout>
```

在布局文件中，分别使用了 3 个按钮来控制音乐的播放、暂停、停止；使用了一个 seekBar 来显示音乐播放的进度；同时在 seekBar 两侧分别使用了 TextView，用来显示音乐已播放了的时长和音乐所有的时长；除此之外，布局文件中还有两个 TextView，分别用于显示播放器名称和音乐文件名称。下面通过修改 MainActivity 来播放音乐。此处需要对 MediaPlayer 的工作流程进行说明。首先需要创建一个 MediaPlayer 对象，然后调用 setDataSource()方法来设置音频文件的路径，再调用 prepare()方法使 MediaPlayer 进入准备状态，接下来调用 start()方法就会开始播放音频，调用 pause()方法就会暂停播放，调用 reset()方法就会停止播放。MainActivity 代码如下：

```
public class MainActivity extends AppCompatActivity implements View.OnClickListener {
    private MediaPlayer mediaPlayer;
    private TextView allTime;
    private TextView playTime;
```

```
    private TextView songName;
    private SeekBar seekBar;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        initView();
        getPermission();
    }

    private void initView() {
        playTime = (TextView) findViewById(R.id.played_time);
        allTime = (TextView) findViewById(R.id.all_time);
        songName = (TextView) findViewById(R.id.song_name);
        seekBar = (SeekBar) findViewById(R.id.seek_bar);

        Button stop = (Button) findViewById(R.id.stop);
        Button play = (Button) findViewById(R.id.play);
        Button pause = (Button) findViewById(R.id.pause);
        stop.setOnClickListener(this);
        play.setOnClickListener(this);
        pause.setOnClickListener(this);

        /**
         * 监听用户对 seekBar 的操作，如果用户滑动，就使用 mediaPlayer 的 seekTo()方法
         */
        seekBar.setOnSeekBarChangeListener(new SeekBar.OnSeekBarChangeListener() {
            @Override
            public void onProgressChanged(SeekBar seekBar, int progress,
                                          boolean fromUser) {
                // fromUser 判断是用户改变的滑块值
                if (fromUser == true) {
                    mediaPlayer.seekTo(progress);
                }
            }
            @Override
            public void onStartTrackingTouch(SeekBar seekBar) {
            }
            @Override
            public void onStopTrackingTouch(SeekBar seekBar) {
            }
        });
```

```
    }

    /**
     * 事实这应该是从其他界面选择要播放的音乐后进入的
     * 这里为了简单起见，直接进入播放界面，而省略了选择步骤
     */
    private void initMediaPlayer() {
        mediaPlayer = new MediaPlayer();
        try {
            //这里应该是从上一个界面传递来的音乐，这里写死
            File file = new File(Environment.getExternalStorageDirectory(),
                    "music/鸽子.mp3");
            // 指定音频文件的路径
            mediaPlayer.setDataSource(file.getPath());
            // 让 MediaPlayer 进入准备状态
            mediaPlayer.prepare();

            //设置显示歌名
            songName.setText(file.getName());
            //设置 seekBar 的最大值
            seekBar.setMax(mediaPlayer.getDuration());
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    @Override
    public void onClick(View v) {
        switch (v.getId()) {
            case R.id.play:
                if (!mediaPlayer.isPlaying()) {
                    // 开始播放
                    mediaPlayer.start();
                    //给进度条设置时长
                    Message message = handler.obtainMessage();
                    message.what = 1;
                    message.arg1 = mediaPlayer.getDuration();
                    handler.sendMessage(message);
                    handler.post(updateThread);
                }
                break;
            case R.id.pause:
                if (mediaPlayer.isPlaying()) {
```

```
                    // 暂停播放
                    mediaPlayer.pause();
                    //取消 seekBar 更新线程
                    handler.removeCallbacks(updateThread);
                }
                break;
            case R.id.stop:
                if (mediaPlayer.isPlaying()) {
                    // 停止播放
                    mediaPlayer.stop();
                    mediaPlayer.reset();
                    initMediaPlayer();
                    //取消 seekBar 更新线程
                    handler.removeCallbacks(updateThread);
                    //将时长等变为零
                    handler.sendEmptyMessage(3);
                }
                break;
            default:
                break;
        }
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
        if (mediaPlayer != null) {
            mediaPlayer.stop();
            mediaPlayer.release();
        }
    }

    private Handler handler = new Handler() {
        @Override
        public void handleMessage(Message msg) {
            super.handleMessage(msg);
            switch (msg.what) {
                case 1:
                    allTime.setText(msg.arg1 / 60000 + ":" + msg.arg1 / 1000 % 60);
                    break;
                case 3:
                    allTime.setText("00:00");
                    playTime.setText("00:00");
```

```
                    seekBar.setProgress(0);
                    break;
            }
        }
    };
    Runnable updateThread = new Runnable() {
        public void run() {
            //获得歌曲现在播放位置并设置成播放进度条的值
            seekBar.setProgress(mediaPlayer.getCurrentPosition());
            playTime.setText(mediaPlayer.getCurrentPosition() / 60000 +
                    ":" + mediaPlayer.getCurrentPosition() / 1000 % 60);
            //每次延迟 100 毫秒再启动线程
            handler.postDelayed(updateThread, 100);
        }
    };

    public void getPermission() {
        if (Build.VERSION.SDK_INT >= 23) {
            int checkPermission = ContextCompat.
                    checkSelfPermission(this, Manifest.permission.READ_EXTERNAL_STORAGE);
            if (checkPermission != PackageManager.PERMISSION_GRANTED) {
                ActivityCompat.requestPermissions(this,
                        new String[]{Manifest.permission.READ_EXTERNAL_STORAGE},
                        111);
                return;
            } else {
                initMediaPlayer();
            }
        } else {
            initMediaPlayer();
        }
    }

    @Override
    public void onRequestPermissionsResult(int requestCode, String[] permissions,
                                           int[] grantResults) {
        switch (requestCode) {
            case 111:
                if (grantResults[0] == PackageManager.PERMISSION_GRANTED) {
                    Toast.makeText(this, "获取权限成功", Toast.LENGTH_SHORT)
                            .show();
                    initMediaPlayer();
                } else {
```

```
                    Toast.makeText(this, "获取权限失败", Toast.LENGTH_SHORT)
                            .show();
                    return;
                }
                break;
            default:
                super.onRequestPermissionsResult(requestCode, permissions, grantResults);
        }
    }
}
```

实现一个最简单的音乐播放器代码必然包括上述代码中的内容，但即使是最简单的播放器，代码量依旧不少。读者在学习时需要借助代码中的注释，非常仔细地阅读。

从代码中可以看出，实例中最先操作的是初始化控件。在这个过程中设置了几个按钮的点击事件，并设置了 SeekBar 的拖动事件，即当再次拖动时就调用 MediaPlayer 的 seekTo()方法，从指定位置开始播放。在初始化控件之后就调用 initMediaPlayer()（在 getPermission()中调用）方法为 MediaPlayer 对象进行初始化操作。在 initMediaPlayer()方法中，首先是通过创建一个 File 对象来指定音频文件的路径，从这里可以看出，我们读取了一个在 music 文件下名为“鸽子.mp3”的音频文件。后面依次调用 setDataSource()方法和 prepare()方法为 MediaPlayer 做好了播放前的准备。同时还设置了歌名，并为 SeekBar 设置了最大值。

当初始化控件和初始化 MediaPlayer 完成之后，就能很清晰地看出，当点击“播放”按钮时会进行判断，如果当前 MediaPlayer 没有正在播放音频，则调用 start()方法开始播放，同时开启一个线程并利用 Handler 机制每隔 100 毫秒更新一次 SeekBar 和已经播放的时长。当点击“暂停”按钮时会判断如果当前 MediaPlayer 正在播放音频就调用 pause()方法暂停播放，同时将更新 SeekBar 的线程移除出 Handler。当点击“停止”按钮时会判断，如果当前 MediaPlayer 正在播放音频，就调用 reset()方法将 MediaPlayer 重置为刚刚创建的状态，然后重新调用一遍 initMediaPlayer()方法，并将更新 SeekBar 的线程移除出 Handler。最后在 onDestroy()方法中分别调用 stop()和 release()方法，将与 MediaPlayer 相关的资源释放掉。

另外，由于这里读取的音乐文件位于应用外部，因此需要读取外部文件的权限。这属于危险权限，所以需要使用 getPermission()方法和 onRequestPermissionsResult()方法来动态获取权限。对于危险权限，需要动态获取和动态获取的步骤相信读者已经很熟悉，这里就不再讲解了。另外，需要说明的是，可能部分读者在使用真机测试时会发现有些手机在 Android 6.0 版本下依旧不需要动态获取权限，这其实是厂商做的深度定制，并不能解决所有手机的问题，也不能应对所有的权限，因此在使用时危险权限必须动态获取。

当然，还需要在 AndroidManifest.xml 中显式地加入读取外部文件的权限：

```
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
```

至此，一个音乐播放器就完成了，运行程序，点击“播放”按钮就可以欣赏宋冬野的“鸽子”了。运行程序，点击“播放”按钮时可以听到音乐，进度条会向前进，已播放时间将增大，暂停时音乐将不再播放，进度条和已播放时间不再变化，点击“停止”按钮时音乐终止播放，

进度条和已播放时间归 0。另外，当拖动进度条时，音乐会从拖到的位置处开始播放，效果如图 11-19 所示。

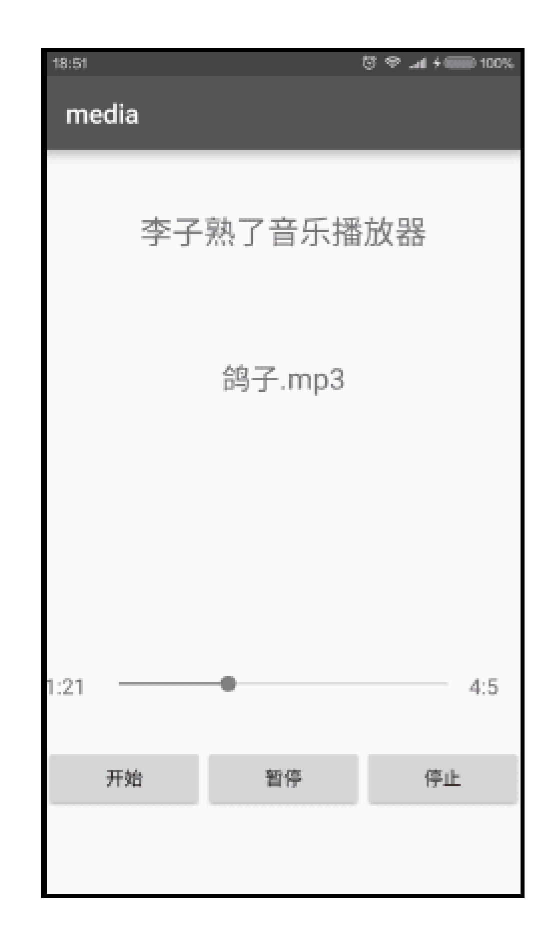


图 11-19　自定义的音乐播放器

这里再讲解一下如何播放在线音乐。其实，播放在线音乐和播放本地音乐只有一个区别，那就是 mediaPlayer.setDataSource(this, uri)方法的参数。在本实例中，如果想要播放在线音乐，只需要将 initMediaPlayer()方法修改如下即可（当然由于不需要读取外部文件，因此动态获取权限的代码可以省去）：

```
private void initMediaPlayer() {
    mediaPlayer = new MediaPlayer();
    //这是李宗盛的“山丘”，读者可以使用本实例进行欣赏
    Uri uri = Uri.parse("http://yinyueshiting.baidu.com/data2/music/" +
            "136340913/1084444261512001 28.mp3?xcode=ea2695ebcdda3c79ed47f60d337fa3fd");
    try {
        mediaPlayer.setDataSource(this, uri);
        mediaPlayer.prepare();
        seekBar.setMax(mediaPlayer.getDuration());
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

如果想要播放网络音乐，获取网络权限则是必不可少的。网络权限属于普通权限，只需要在 AndroidManifest.xml 文件中显式声明即可：

```
<uses-permission android:name="android.permission.INTERNET"/>
```

不管是使用本地音乐还是播放网络音乐，运行之后的效果都是一样的。

11.5.2　开发一个视频器

开发一个视频播放器的方法有 3 种，最简单的方法是使用其自带的播放器，这种方法只

需使用一个 Action 为 ACTION_VIEW、Data 为要播放视频的 Uri、Type 为其 MIME 类型的意图去打开系统播放器即可，这种方式灵活度过低，一般不使用。第二种方法是使用 MediaPlayer 类和 SurfaceView 来实现，这种方式很灵活，可定制化程度高，使用较为广泛，但是开发难度相对较高。第三种方法是使用 VideoView 来播放，VideoView 只是对 MediaPlayer 做了一个很好的封装，它的背后仍然是使用 MediaPlayer 来对视频文件进行控制的。相对于使用 MediaPlayer 来进行播放，VideoView 虽然操纵简单，但是在视频格式的支持以及播放效率方面都存在较大的不足。虽然如此，VideoView 还是可以满足大部分视频播放的需求，而且 VideoView 的大部分 API 和 MediaPlayer 很相似，学会了 VideoView，再去使用 MediaPlayer 就会容易一些，所以在这里我们将使用 VideoView 来进行视频播放器的开发。

就像前面说的 VideoView 是对 MediaPlayer 的封装，所以常用方法比较类似，如表 11-3 所示。

表11-3　Android中常用的控件类

方法名	方法描述
setVideoPath()	设置要播放的视频文件
start()	开始或继续播放视频，对应 MediaPlayer 的 start()方法
pause()	暂停播放视频，对应 MediaPlayer 的 start()方法
resume()	将视频从头开始播放，对应 MediaPlayer 的 pause()方法
seekTo()	从指定的位置开始播放视频，对应 MediaPlayer 的 seekTo()方法
isPlaying()	判断当前是否正在播放视频，对应 MediaPlayer 的 isPlaying()方法
getDuration()	获取载入的视频文件时长，对应 MediaPlayer 的 getDuration()方法
stopPlayback()	停止播放视频，对应 MediaPlayer 的 stop()加上 release()方法
suspend()	将 VideoView 挂起，对应 MediaPlayer 的 release(false)

下面通过实例来说明如何使用这些方法构建视频播放器。创建一个新的项目，并修改 activity_main.xml 文件：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context="com.buaa.video.MainActivity">

    <VideoView
        android:id="@+id/video"
        android:layout_width="match_parent"
        android:layout_height="400dp" />

    <LinearLayout
        android:layout_width="match_parent"
```

```
        android:layout_height="wrap_content"
        android:layout_gravity="center_vertical"
        android:orientation="horizontal">

        <TextView
            android:id="@+id/played_time"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="00:00"
            android:textSize="16dp" />

        <SeekBar
            android:id="@+id/seek_bar"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_weight="7" />

        <TextView
            android:id="@+id/all_time"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="00:00"
            android:textSize="16dp" />
    </LinearLayout>

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent">

        <Button
            android:id="@+id/play"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="播放" />

        <Button
            android:id="@+id/pause"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_weight="1"
```

```
                android:text="暂停" />

            <Button
                android:id="@+id/stop"
                android:layout_width="0dp"
                android:layout_height="wrap_content"
                android:layout_weight="1"
                android:text="停止" />
        </LinearLayout>
    </LinearLayout>
```

这里有一个新的控件 VideoView，是专门用于显示视频的。除此之外，还有 3 个按钮、两个 TextView 以及一个 SeekBar。这些读者应该很熟悉，因为它们与音频播放器中的布局是一样的，当然作用也是一样的。下面通过修改 MainActivity 来完成具体的播放逻辑操作。前面我们说 VideoView 是对 MediaPlayer 的封装，其 API 具有相似性，必然 MainActivity 中逻辑部分的代码也将具有相似性，代码如下：

```
public class MainActivity extends AppCompatActivity implements View.OnClickListener {
    private VideoView videoView;
    private Button play;
    private Button pause;
    private Button stop;
    private SeekBar seekBar;
    private TextView playTime;
    private TextView allTime;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        initView();
        getPermission();
    }

    private void initView() {
        playTime = (TextView) findViewById(R.id.played_time);
        allTime = (TextView) findViewById(R.id.all_time);

        play = (Button) findViewById(R.id.play);
        pause = (Button) findViewById(R.id.pause);
        stop = (Button) findViewById(R.id.stop);
        videoView = (VideoView) findViewById(R.id.video);
        play.setOnClickListener(this);
        pause.setOnClickListener(this);
        stop.setOnClickListener(this);
```

```
        seekBar = (SeekBar) findViewById(R.id.seek_bar);
        seekBar.setOnSeekBarChangeListener(new SeekBar.OnSeekBarChangeListener() {
            @Override
            public void onProgressChanged(SeekBar seekBar, int progress,
                                          boolean fromUser) {
                if (fromUser == true) {
                    videoView.seekTo(progress);
                }
            }
            @Override
            public void onStartTrackingTouch(SeekBar seekBar) {}
            @Override
            public void onStopTrackingTouch(SeekBar seekBar) {}
        });
    }

    private void initVideoPath() {
        File file = new File(Environment.getExternalStorageDirectory(),
                "music/爱的代价.mp4");
        videoView.setVideoPath(file.getPath());
    }

    @Override
    public void onClick(View v) {
        switch (v.getId()) {
            case R.id.play:
                if (!videoView.isPlaying()) {
                    videoView.start();
                    seekBar.setMax(videoView.getDuration());
                    Message message = handler.obtainMessage();
                    message.what = 1;
                    message.arg1 = videoView.getDuration();
                    handler.sendMessage(message);
                    handler.post(updateThread);
                }
                break;
            case R.id.pause:
                if (videoView.isPlaying()) {
                    videoView.pause();
                    handler.removeCallbacks(updateThread);
                }
                break;
```

```
                case R.id.stop:
                    if (videoView.isPlaying()) {
                        videoView.stopPlayback();
                        initVideoPath();
                        handler.removeCallbacks(updateThread);
                        handler.sendEmptyMessage(3);
                    }
                    break;
            }
        }

        @Override
        protected void onDestroy() {
            super.onDestroy();
            if (videoView != null) {
                videoView.suspend();
            }
        }

        private Handler handler = new Handler() {
            @Override
            public void handleMessage(Message msg) {
                super.handleMessage(msg);
                switch (msg.what) {
                    case 1:
                        allTime.setText(msg.arg1 / 60000 + ":" + msg.arg1 / 1000 % 60);
                        break;
                    case 3:
                        allTime.setText("00:00");
                        playTime.setText("00:00");
                        seekBar.setProgress(0);
                        break;
                }
            }
        };
        Runnable updateThread = new Runnable() {
            public void run() {
                seekBar.setProgress(videoView.getCurrentPosition());
                playTime.setText(videoView.getCurrentPosition() / 60000 +
                        ":" + videoView.getCurrentPosition() / 1000 % 60);
                handler.postDelayed(updateThread, 100);
            }
        };
```

```
    public void getPermission() {
        if (Build.VERSION.SDK_INT >= 23) {
            int checkPermission = ContextCompat.
                    checkSelfPermission(this, Manifest.permission.READ_EXTERNAL_STORAGE);
            if (checkPermission != PackageManager.PERMISSION_GRANTED) {
                ActivityCompat.requestPermissions(this,
                        new String[]{Manifest.permission.READ_EXTERNAL_STORAGE},
                        111);
                return;
            } else {
                initVideoPath();
            }
        } else {
            initVideoPath();
        }
    }

    @Override
    public void onRequestPermissionsResult(int requestCode, String[] permissions,
                                           int[] grantResults) {
        switch (requestCode) {
            case 111:
                if (grantResults[0] == PackageManager.PERMISSION_GRANTED) {
                    Toast.makeText(this, "获取权限成功", Toast.LENGTH_SHORT)
                            .show();
                    initVideoPath();
                } else {
                    Toast.makeText(this, "获取权限失败", Toast.LENGTH_SHORT)
                            .show();
                    return;
                }
                break;
            default:
                super.onRequestPermissionsResult(requestCode, permissions, grantResults);
        }
    }
}
```

看到上面的代码，读者可能会非常惊奇。除了部分地方外，这与使用 MediaPlayer 进行音频播放的代码简直是一模一样的。造成这种相似性的原因是 VideoView 是对 MediaPlayer 的封装。代码中少有的不同点在于获取 VideoView 的方式、获取视频的方式、停止视频、清理资源的方式不同。获取 VideoView 的实例是通过 Context.findViewById()方法来实现的，然后调

用 setVideoPath(String path)方法来加载资源。其他的不同之处也大体与 MediaPlayer 相似，这里不再讲解。

与读取音频相同，这里也需要在 AndroidManifest.xml 中显式地声明读取外部文件的权限：

```
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
```

至此，一个视频播放器就完成了，运行程序，点击“播放”按钮就可以欣赏李宗盛的 MV“爱的代价”了。运行程序，点击“播放”按钮时就能观看视频，进度条会向前进，已播放时间将增大，“暂停”按钮时视频将不再播放，进度条和已播放时间不再变化，点击“停止”按钮时视频终止播放，进度条和已播放时间归零。另外，当拖动进度条时，视频会从拖到的位置处开始播放，效果如图 11-20 所示。



图 11-20　自定义的视频播放器

视频播放器也可以播放在线视频。这里需要将 initVideoPath()方法修改如下：

```
private void initVideoPath() {
    Uri uri = Uri.parse("网络视频地址");
    videoView.setVideoURI(uri);
}
```

当然，播放网络视频还需要添加网络权限，相信读者对此已经不再陌生，就不再重复演示了。

11.6　录视频与录音频

11.5 节讲解了如何使用系统提供的 API 去实现音频播放器和视频播放器，并播放了本地音视频和网络音视频。这些音视频都是已有的，除此之外，我们还可以通过手机录制音视频。在 Android 中，也提供了录制音视频的 API，下面我们分别对录音频和录视频进行讲解。

11.6.1　录制音频

Android 中音频的录制可以通过 MediaRecorder 类或者 AudioRecorder 类来完成。MediaRecorder 本是多媒体录制控件，可以同时录制视频和语音，当不指定视频源时就只录制语音；AudioRecorder 只能录制语音。两者录制的区别在于，MediaRecorder 固定了语音的编码格式，而且使用时指定输出文件，在录制的同时系统将语音数据写入文件；AudioRecorder 输出的是 pcm，即原始音频数据，使用者需要自己读取这些数据，这样的好处是可以根据需要边录制边对音频数据进行处理，读取的同时也可以保存到文件进行存储。

这里我们使用 MediaRecorder 类来进行音频的录制。一般情况下，使用 MediaRecorder 类录制一个音频需要下面 8 个步骤：

步骤 01　创建 MediaRecorder 对象，直接使用 new MediaRecorder()即可。

步骤02 调用 MediaRecorder 对象的 setAudioSource()方法设置声音来源，一般传入 MediaRecorder. AudioSource.MIC 参数指定录制来自麦克风的声音。

步骤03 调用 MediaRecorder 对象的 setOutputFormat()设置所录制的音频文件的格式。

步骤04 调用 MediaRecorder 对象的 setAudioEncoder()、setAudioEncodingBitRate(intbitRate)、setAudioSamplingRate(int samplingRate)设置所录制的声音的编码格式、编码位率、采样率等。这些参数将可以控制所录制的声音的品质、文件的大小。一般来说，声音品质越好，声音文件越大。

步骤05 调用 MediaRecorder 的 setOutputFile(Stringpath)方法设置录制的音频文件的保存位置。

步骤06 调用 MediaRecorder 的 prepare()方法准备录制。

步骤07 调用 MediaRecorder 对象的 start()方法开始录制。

步骤08 录制完成之后，调用 MediaRecorder 对象的 stop()方法停止录制，并调用 release()方法释放资源。

下面通过实例来进行说明。新建一个项目，修改 activity_main.xml 的布局文件：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:gravity="center"
    android:orientation="vertical"
    tools:context="com.buaa.record_sound.MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="0dp"
        android:layout_weight="2"
        android:paddingTop="45dp"
        android:text="李子熟了录音机"
        android:textSize="24sp" />

    <TextView
        android:id="@+id/record_state"
        android:layout_width="wrap_content"
        android:layout_height="0dp"
        android:layout_weight="5"
        android:paddingTop="45dp"
        android:text="准备录音"
        android:textSize="22sp" />

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="0dp"
```

```
            android:layout_weight="2"
            android:orientation="horizontal">

            <Button
                android:id="@+id/start"
                android:layout_width="0dp"
                android:layout_height="wrap_content"
                android:layout_weight="1"
                android:text="开始" />

            <Button
                android:id="@+id/stop"
                android:layout_width="0dp"
                android:layout_height="wrap_content"
                android:layout_weight="1"
                android:text="停止" />
        </LinearLayout>
    </LinearLayout>
```

经过这么多章节的讲解，相信读者对这样的布局文件已经很容易理解了。这里使用了两个控件来分别触发开始录音和停止录音，并使用一个 TextView 来告知读者系统正在录音或者录音结束了。下面修改 Activity 代码来实现录音的逻辑，代码如下：

```
public class MainActivity extends AppCompatActivity implements View.OnClickListener {
    private Button start;
    private TextView recordState;
    private Button stop;
    private MediaRecorder mediaRecorder;
    private File outFile;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        initView();
        getPermission();
    }

    private void initView() {
        recordState = (TextView) findViewById(R.id.record_state);
        start = (Button) findViewById(R.id.start);
        stop = (Button) findViewById(R.id.stop);
        start.setOnClickListener(this);
        stop.setOnClickListener(this);
```

```
    }

    @Override
    public void onClick(View v) {
        switch (v.getId()) {
            case R.id.start:
                try {
                    // 创建保存录音的音频文件
                    outFile = new File(Environment
                            .getExternalStorageDirectory(), "music/" +
                            System.currentTimeMillis() + ".amr");
                    mediaRecorder = new MediaRecorder();
                    // 设置录音的声音来源
                    mediaRecorder.setAudioSource(MediaRecorder
                            .AudioSource.MIC);
                    // 设置录制的声音的输出格式（必须在设置声音编码格式之前设置）
                    mediaRecorder.setOutputFormat(MediaRecorder
                            .OutputFormat.AMR_NB);
                    // 设置声音编码的格式
                    mediaRecorder.setAudioEncoder(MediaRecorder
                            .AudioEncoder.AMR_NB);
                    mediaRecorder.setOutputFile(outFile.getAbsolutePath());
                    mediaRecorder.prepare();
                    // 开始录音
                    mediaRecorder.start();
                    recordState.setText("正在录音中......");
                    start.setEnabled(false);
                    stop.setEnabled(true);
                } catch (IOException e) {
                    e.printStackTrace();
                }
                break;
            case R.id.stop:
                if (outFile != null && outFile.exists()) {
                    // 停止录音
                    mediaRecorder.stop();
                    // 释放资源
                    mediaRecorder.release();
                    mediaRecorder = null;
                    recordState.setText("录音结束......");
                    start.setEnabled(true);
                    stop.setEnabled(false);
                }
```

```
                break;
            }
        }

        public void getPermission() {
            if (Build.VERSION.SDK_INT >= 23) {
                int checkPermission1 = ContextCompat.
                        checkSelfPermission(this,
                                Manifest.permission.READ_EXTERNAL_STORAGE);
                int checkPermission2 = ContextCompat.
                        checkSelfPermission(this,
                                Manifest.permission.WRITE_EXTERNAL_STORAGE);
                int checkPermission3 = ContextCompat.
                        checkSelfPermission(this,
                                Manifest.permission.RECORD_AUDIO);
                if (checkPermission1 != PackageManager.PERMISSION_GRANTED
                        || checkPermission2 != PackageManager.PERMISSION_GRANTED
                        || checkPermission3 != PackageManager.PERMISSION_GRANTED) {
                    ActivityCompat.requestPermissions(this,
                            new String[]{
                                    Manifest.permission.READ_EXTERNAL_STORAGE,
                                    Manifest.permission.WRITE_EXTERNAL_STORAGE,
                                    Manifest.permission.RECORD_AUDIO
                            }, 111);
                    return;
                }
            }
        }

        @Override
        public void onRequestPermissionsResult(
                int requestCode, String[] permissions, int[] grantResults) {
            switch (requestCode) {
                case 111:
                    if (grantResults[0] == PackageManager.PERMISSION_GRANTED) {
                        Toast.makeText(this, "获取权限成功",
                                Toast.LENGTH_SHORT).show();
                    } else {
                        Toast.makeText(this, "获取权限失败",
                                Toast.LENGTH_SHORT).show();
                    }
                    break;
                default:
```

```
                    super.onRequestPermissionsResult(requestCode, permissions, grantResults);
                }
            }
        }
```

这里完全是按照上面所描述的 8 个步骤来进行录音操作的，所以就不再进行讲解了。由于向外部存储读取和写入数据以及使用系统麦克风都是危险权限，因此这里需要动态获取这些权限。完成这些之后，在 AndroidManifest.xml 中显式添加权限声明：

```
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.RECORD_AUDIO" />
```

运行程序，点击“开始”按钮就可以录音了，此时界面中的 TextView 会显示出“正在录音”字样。当点击“停止”按钮后，录音结束，此时界面中的 TextView 会显示出“录音结束”字样。录音结束后，打开在代码中设置的路径，即根目录下 music 文件夹，就可以找到刚才的录音，使用播放器就可以播放它。效果如图 11-21 所示：

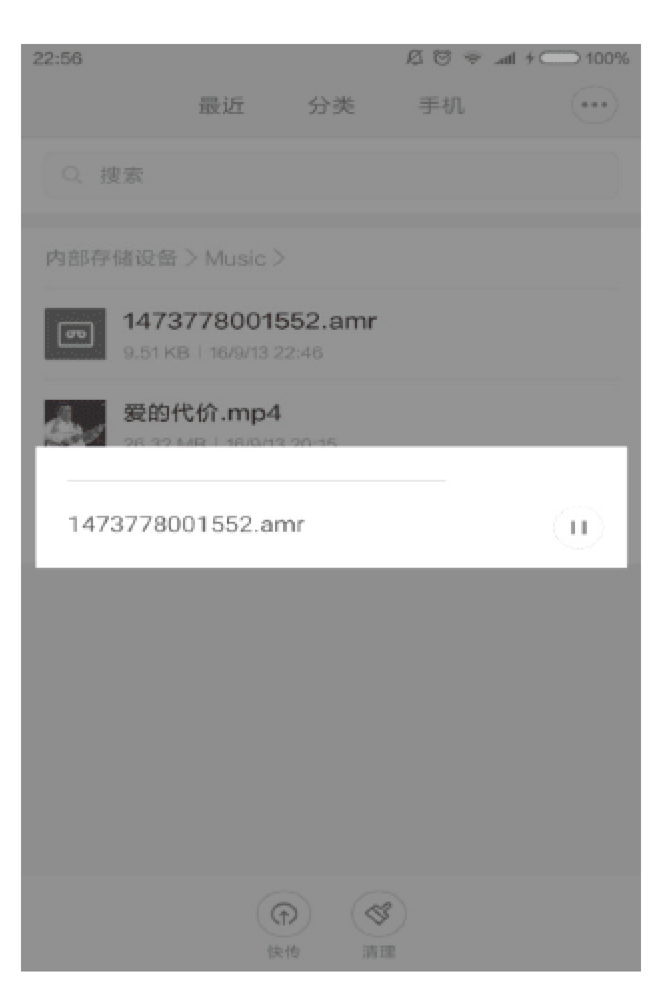


图 11-21　录制音频并播放

讲到这里，不知道读者是否还记得在 11.5 节开发的音频播放器，在该程序中是将音频文件写死在程序中的。现在学会了录制音频，可以将两者整合到一个程序中，当录制完音频之后，使用音频播放器进行播放。方法很简单，将两个 Activity 放置在一个项目中，只需要在录音完成时跳转到音频播放器的 Activity 中，并将录制的音频路径作为参数传递过去。然后在音频播放器的 Activity 中接收音频路径，并使用 MediaPlayer 播放该路径下的音频文件。有兴趣的读者可以进行尝试，这里就不再演示了。

11.6.2　录制视频

在 Android 系统中录制视频也是使用 MediaRecorder 类。与录制音频相比，录制视频的步骤要多一些，具体来说有如下几步：

步骤 01 创建 MediaRecorder 对象，直接使用 new MediaRecorder()即可。

步骤 02 调用 MediaRecorder 对象的 setVideoSource()方法设置视频的来源，一般传入 MediaRecorder.VideoSource.CAMERA 参数指定录制来自摄像头的图像。

步骤 03 调用 MediaRecorder 对象的 setAudioSource()方法设置声音来源，一般传入 MediaRecorder.AudioSource.MIC 参数指定录制来自麦克风的声音。

步骤 04 调用 MediaRecorder 对象的 setOutputFormat()设置录制音频文件的格式。

步骤 05 调用 MediaRecorder 对象的 setVideoEncoder 设置录制的视频编码格式等。这些参数可以控制所录制的视频品质，文件大小，一般视频品质越好，视频文件越大。

步骤 06 调用 MediaRecorder 对象的 setAudioEncoder、setAudioEncodingBitRate(int)、setAudioSamplingRate(int)设置录制声音的编码格式、编码位率、采样率等。

步骤 07 调用 setVideoFrameRate(20)设置录制的视频帧率，必须放在设置编码和格式的后面，否则报错。

步骤 08 调用 setVideoSize(176, 144)，设置视频录制的分辨率，必须放在设置编码和格式的后面，否则报错。

步骤 09 调用 setPreviewDisplay(sv.getHolder().getSurface())，这是视频的预览效果。

步骤 10 调用 MediaRecorder 对象的 setOutputFile(String path)设置录制文件保存的位置。

步骤 11 调用 MediaRecorder 的 prepare()方法准备录制。

步骤 12 调用 MediaRecorder 对象的 start()方法开始录制。

步骤 13 录制完成后，调用 MediaRecorder 对象的 stop()方法停止录制，并调用 release()方法释放资源。

下面通过实例来进行说明。新建一个项目，修改 activity_main.xml 的布局文件：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:gravity="center"
    android:orientation="vertical"
    tools:context="com.buaa.record_sound.MainActivity">

    <SurfaceView
        android:id="@+id/record_video"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1" />

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal">

        <Button
            android:id="@+id/start"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="录制" />

        <Button
            android:id="@+id/stop"
            android:layout_width="0dp"
```

```
                android:layout_height="wrap_content"
                android:layout_weight="1"
                android:text="停止" />
        </LinearLayout>
    </LinearLayout>
```

这里使用了一个 SurfaceView 控件，用于预览录制的视频。布局中还加入了两个按钮，分别用于开始录制视频和停止录制视频。下面修改 MainActivity 代码来实现录制视频的逻辑，代码如下：

```
public class MainActivity extends AppCompatActivity implements View.OnClickListener {

    private File videoFile;
    private MediaRecorder mediaRecorder;
    private Button start;
    private Button stop;
    private SurfaceView recordSurface;

    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        intiView();
        getPermission();
    }

    private void intiView() {
        // 设置横屏显示
        setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE);
        // 设置全屏
        getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,
                WindowManager.LayoutParams.FLAG_FULLSCREEN);
        // 选择支持半透明模式，在有 surfaceview 的 activity 中使用
        getWindow().setFormat(PixelFormat.TRANSLUCENT);
        // 获取程序界面中的两个按钮
        start = (Button) findViewById(R.id.start);
        stop = (Button) findViewById(R.id.stop);
        stop.setEnabled(false);
        // 为两个按钮的单击事件绑定监听器
        start.setOnClickListener(this);
        stop.setOnClickListener(this);
        // 获取程序界面中的 SurfaceView
        recordSurface = (SurfaceView) this.findViewById(R.id.record_video);
        // 设置分辨率
        recordSurface.getHolder().setFixedSize(1280, 720);
```

```
        // 设置该组件让屏幕不会自动关闭
        recordSurface.getHolder().setKeepScreenOn(true);
    }

    @Override
    public void onClick(View v) {
        switch (v.getId()) {
            // 单击录制按钮
            case R.id.start:
                try {
                    // 创建保存录制视频的视频文件
                    videoFile = new File(Environment
                            .getExternalStorageDirectory()
                            , "music/" + System.currentTimeMillis() + ".mp4");
                    // 创建 MediaPlayer 对象
                    mediaRecorder = new MediaRecorder();
                    mediaRecorder.reset();
                    // 设置从麦克风采集声音(或来自录像机的声音 AudioSource.CAMCORDER)
                    mediaRecorder.setAudioSource(MediaRecorder
                            .AudioSource.MIC);
                    // 设置从摄像头采集图像
                    mediaRecorder.setVideoSource(MediaRecorder
                            .VideoSource.CAMERA);
                    // 设置视频文件的输出格式
                    // 必须在设置声音编码格式、图像编码格式之前设置
                    mediaRecorder.setOutputFormat(MediaRecorder
                            .OutputFormat.THREE_GPP);
                    // 设置声音编码的格式
                    mediaRecorder.setAudioEncoder(MediaRecorder
                            .AudioEncoder.AMR_NB);
                    // 设置图像编码的格式
                    mediaRecorder.setVideoEncoder(MediaRecorder
                            .VideoEncoder.H264);
                    mediaRecorder.setVideoSize(1280, 720);
                    // 每秒 4 帧
                    mediaRecorder.setVideoFrameRate(20);
                    mediaRecorder.setOutputFile(videoFile.getAbsolutePath());
                    // 指定使用 SurfaceView 来预览视频
                    mediaRecorder.setPreviewDisplay(recordSurface
                            .getHolder().getSurface());
                    mediaRecorder.prepare();
                    // 开始录制
                    mediaRecorder.start();
```

```
                //让录制按钮不可用、停止按钮可用
                start.setEnabled(false);
                stop.setEnabled(true);
            } catch (Exception e) {
                e.printStackTrace();
            }
            break;
        // 单击停止按钮
        case R.id.stop:
            // 停止录制
            mediaRecorder.stop();
            // 释放资源
            mediaRecorder.release();
            mediaRecorder = null;
            //让录制按钮可用，停止按钮不可用
            start.setEnabled(true);
            stop.setEnabled(false);
            break;
    }
}

public void getPermission() {
    if (Build.VERSION.SDK_INT >= 23) {
        int checkPermission1 = ContextCompat.
                checkSelfPermission(this,
                        Manifest.permission.READ_EXTERNAL_STORAGE);
        int checkPermission2 = ContextCompat.
                checkSelfPermission(this,
                        Manifest.permission.WRITE_EXTERNAL_STORAGE);
        int checkPermission3 = ContextCompat.
                checkSelfPermission(this,
                        Manifest.permission.RECORD_AUDIO);
        int checkPermission4 = ContextCompat.
                checkSelfPermission(this,
                        Manifest.permission.CAMERA);
        if (checkPermission1 != PackageManager.PERMISSION_GRANTED
                || checkPermission2 != PackageManager.PERMISSION_GRANTED
                || checkPermission3 != PackageManager.PERMISSION_GRANTED
                || checkPermission4 != PackageManager.PERMISSION_GRANTED) {
            ActivityCompat.requestPermissions(this,
                    new String[]{
                            Manifest.permission.READ_EXTERNAL_STORAGE,
                            Manifest.permission.WRITE_EXTERNAL_STORAGE,
```

```
                                                  Manifest.permission.RECORD_AUDIO,
                                                  Manifest.permission.CAMERA
                                        }, 111);
                              return;
                         }
                    }
               }

               @Override
               public void onRequestPermissionsResult(
                         int requestCode, String[] permissions, int[] grantResults) {
                    switch (requestCode) {
                         case 111:
                              if (grantResults[0] == PackageManager.PERMISSION_GRANTED) {
                                   Toast.makeText(this, "获取权限成功",
                                             Toast.LENGTH_SHORT).show();
                              } else {
                                   Toast.makeText(this, "获取权限失败",
                                             Toast.LENGTH_SHORT).show();
                              }
                              break;
                         default:
                              super.onRequestPermissionsResult(requestCode, permissions, grantResults);
                    }
               }
          }
```

这里的代码与录音程序比较相似，都是严格按照步骤来进行录制，所以不再进行过多讲解。这里需要读写外部存储，以及调用相机、麦克风，这些都是危险权限，所以需要动态获取这 4 个权限。完成这些之后，在 AndroidManifest.xml 中显式添加权限声明：

```
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.RECORD_AUDIO" />
<uses-permission android:name="android.permission.CAMERA" />
```

运行程序，允许应用获取相关权限之后点击“录制”按钮就可以录制视频了，此时“录制”按钮会进入不可点击状态，然后点击“停止”按钮，视频就录制完成了，而此时“录制”按钮重新变得可用，即可再次录制视频了，如图 11-22 所示。



图 11-22　录制视频

此时视频文件已经保存在 music 文件夹下了，点击播放，如图 11-23 所示。

图 11-23　播放录制的视频

11.6.1 小节说可以将录音和音频播放器两个实例整合到一起，其实录视频和视频播放器也可以整合到一起。方法同样很简单，将两个 Activity 放置在一个项目中，只需要在视频录制完成时跳转到视频播放器的 Activity 中，并将视频的路径作为参数传递过去。然后在视频播放器的 Activity 中接收该视频路径并播放视频。有兴趣的读者可以进行尝试。

11.7　小　　结

本章我们主要对 Android 系统中的各种多媒体技术进行了学习，其中包括通知的使用技巧、调用摄像头拍照、从相册中选取照片、播放音频和视频文件，以及如何进行视频和音频的录制。此外，我们还学习了如何使用 Android 提供的 API 来接收、发送和拦截短信，这使得我们甚至可以编写一个自己的短信程序来替换系统的短信程序。本章的内容非常多，而且这些技术在实际开发中也经常使用，希望读者务必多花时间将它们消化掉。



第12章 传感器与地理位置定位

在 Android 中除了前面章节中所讲到的一些基本开发技术外，还有一些特色开发技术。这些技术最具代表性，是地理信息服务以及传感器技术。使用这些技术可以开发出更多且好玩的应用。本章的学习重点就是传感器与地理信息服务这两种 Android 中最具特色的开发技术。另外，由于传感器与地理信息技术不能在模拟器上运行，因此建议使用真机。

12.1 传 感 器

12.1.1 传感器简介

传感器是一种微型的物理设备，能够探测、感受到外界的信号，并按一定规律转换成我们所需要的信息。大多数 Android 设备有内置的传感器，并被用于测量运动、方向和各种环境条件。这些传感器能提供高精度和准确的原始数据，我们可以使用这些传感器监控设备三维运动或者位置，测量设备周围的环境变化，推断用户的复杂手势（摇一摇原理），例如倾斜、震动、旋转或者振幅。同样的，天气应用可能使用设备的温度传感器和湿度传感器的数据来计算和报告结露点，或者旅行应用可能使用磁场传感器和加速度传感器来报告一个指南针方位。Android 系统支持十余种传感器的类型，细分起来可以分为 3 大类：

- 位移传感器，沿 3 个轴线测量加速度和旋转，这类传感器包含加速度、重力、矢量传感器和陀螺仪。
- 环境传感器，测量各种环境参数，例如周围的空气温度和压力、光线以及湿度。这类传感器包含气压、光线和温度传感器。
- 位置传感器，测量设备的物理位置。这类传感器包含方向和磁力传感器。

要进行传感器的开发，主要使用 android.hardware 包下的 Sensor 类、SensorEvent 类、SensorManager 类以及一个 SensorEventListener 接口。SensorManager 负责传感器管理的工作，负责注册监听某 Sensor 的状态；Sensor 的数据通过 SensorEvent 返回。Android 中每个传感器的开发都比较相似，其一般开发的模式如下：

```
public class MainActivity extends AppCompatActivity implements SensorEventListener {
    private SensorManager sensorManager;
    private Sensor sensor;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        getSensorManager();
    }
    private void getSensorManager() {
        sensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
        /**
         * 传入参数决定传感器类型
         * Sensor.TYPE_ACCELEROMETER：加速度传感器
         * Sensor.TYPE_LIGHT:光照传感器
         * Sensor.TYPE_GRAVITY:重力传感器
         * SensorManager.getOrientation():方向传感器
```

```
             */
            sensor = sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
        }
        @Override
        protected void onResume() {
            super.onResume();
            if (sensorManager != null) {
                //一般在 onResume 方法中进行注册
                /**
                 * 第三个参数决定传感器信息更新速度
                 * SensorManager.SENSOR_DELAY_NORMAL：一般
                 * SENSOR_DELAY_FASTEST：最快
                 * SENSOR_DELAY_GAME：比较快，适合游戏
                 * SENSOR_DELAY_UI：慢
                 */
                sensorManager.registerListener(this, sensor,
                        SensorManager.SENSOR_DELAY_NORMAL);
            }
        }
        @Override
        protected void onPause() {
            super.onPause();
            if (sensorManager != null) {
                //解除注册
                sensorManager.unregisterListener(this, sensor);
            }
        }
        @Override
        public void onSensorChanged(SensorEvent event) {
            //Sensor 发生变化时，在此通过 event.values 获取数据
        }
        @Override
        public void onAccuracyChanged(Sensor sensor, int accuracy) {
            //注册的 Sensor 精度发送变化时，在此处处理
        }
    }
```

代码中的注释很详细，简单来说，就是在 onCreate()方法中获取 SensorManager 对象，然后通过 SensorManager 获取相应的 Sensor。然后在 onResume()方法中进行注册，并在 onPause()方法中解除注册。同时使用 SensorEventListener 接口对 Sensor 进行监听。监听的回调方法有两个，onAccuracyChanged()方法用于处理传感器的精度发生变化的逻辑，onSensorChanged()方法用于处理 Sensor 的值发生变化时的逻辑。

12.1.2 加速度传感器

这里我们将通过一个“摇一摇”应用来说明如何使用加速度传感器。但在开发之前，我们先对加速度传感器进行一些简单的介绍。

加速度是一种用于描述物体运动速度改变快慢的物理量，以 m/s^2 为单位。在静止时，加速度传感器返回的值为地表上静止物体的重力加速度，约为 $9.8m/s^2$。加速度传感器输出的信息存放在 SensorEvent 的 values 数组中的，此时的 values 数组中会有 3 个值，分别代表手机在 *x* 轴、*y* 轴和 *z* 轴方向上的加速度信息。*x* 轴、*y* 轴、*z* 轴在空间坐标系上的含义如图 12-1 所示。

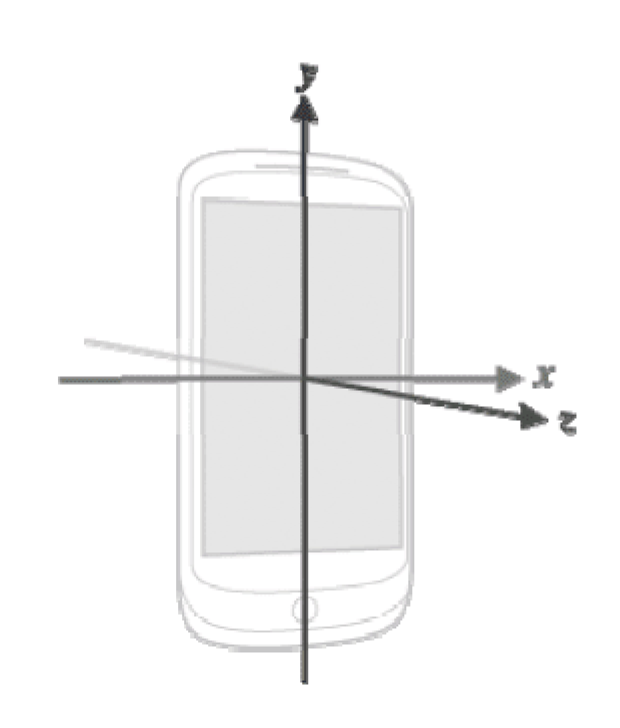


图 12-1 空间坐标系在手机上的含义

根据力学原理，我们知道重力的作用永远是向下的，所以当手机竖直时，重力作用在 *y* 轴，平放在 *z* 轴，横立则在 *x* 轴。根据输出的值和手机的空间坐标系以及重力的作用原理就可以判断出手机放置的状态，比如，当 *x* 轴的值接近重力加速度或者接近负的重力加速度时，说明设备处于横立状态，同时值为正时左边朝下，值为负时右边朝下。

如果要开发一个“摇一摇”应用，根据静止时的加速度约为 $9.8m/s^2$ 以及摇动时加速度会发生变化，我们可以设置一个加速度的上限，比如说 $20m/s^2$，当达到这个上限时就认为手机进行了摇动。了解了其中的原理之后，开发这个应用很简单，只需要修改前面模板中的传感器类型为加速度传感器，即

```
sensor = sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
```

然后，修改 onSensorChanged()即可：

```
@Override
public void onSensorChanged(SensorEvent event) {
    float x = event.values[0];
    float y = event.values[0];
    float z = event.values[0];
    if(x>20||y>20||z>20){
        Toast.makeText(this,"欢迎使用摇一摇",Toast.LENGTH_LONG).show();
    }
}
```

运行程序，摇动手机，界面就会出现一条 Toast，以显示提示信息。

12.1.3 光线传感器

光线传感器可以用来感知周围的光线环境变化。借助这个原理，我们可以开发出一个光线探测器。有了 12.1.2 小节的学习，开发光线探测器就非常简单了，只需要修改前面模板中的传感器类型为光线传感器即可：

```
sensor = sensorManager.getDefaultSensor(Sensor.TYPE_LIGHT);
```

然后，修改 onSensorChanged()：

```
@Override
public void onSensorChanged(SensorEvent event) {
    float light = event.values[0];
    Log.i("Light", "当前光线强度为：" + light + "勒克斯");
}
```

event.values 返回的数组的第一个值就是光线强度值，单位是勒克斯。勒克斯是照度的国际单位（SI），又称米烛光，1 流明的光通量均匀分布在 1 平方米面积上的照度就是一勒克斯，可以标作勒[克斯]，简称勒，简作 lx。适宜于阅读的照度约为 600 勒克斯，如果此时你在阅读本书或者在编写程序，那么运行此程序，在控制台上观察周围的光线强度，看看是否适合阅读。笔者的手机最初是放置在书桌旁边的，此时的光线显然是适宜工作的。如果是阴天，当将手机放置到屋外时，光线强度会瞬间变得很低，不再适合阅读和其他工作。当将手机放置到抽屉时，光线强度很快就会变成 0 勒克斯。效果如下：

```
7194-7194/com.buaa.sensor I/Light: 当前光线值为：776.568勒克斯
7194-7194/com.buaa.sensor I/Light: 当前光线值为：42.705994勒克斯
7194-7194/com.buaa.sensor I/Light: 当前光线值为：50.304勒克斯
7194-7194/com.buaa.sensor I/Light: 当前光线值为：53.447998勒克斯
7194-7194/com.buaa.sensor I/Light: 当前光线值为：48.207993勒克斯
7194-7194/com.buaa.sensor I/Light: 当前光线值为：61.046005勒克斯
7194-7194/com.buaa.sensor I/Light: 当前光线值为：2.3580017勒克斯
7194-7194/com.buaa.sensor I/Light: 当前光线值为：0.0勒克斯
```

方向传感器

在 Android 中方向传感器的使用场景要比其他的传感器更为广泛，因为它能够准确地判断出手机在各个方向的旋转角度，利用这些角度就可以编写出指南针、地平仪等有用的工具。

方向传感器的使用相较于其他传感器有所不同。虽然在 API 中有 TYPE_ORIENTATION 常量，可以像得到加速度传感器那样得到方向传感器 Sensor.TYPE_ORIENTATION，但是这样做的话，在新版的 SDK 中就会提示"这种方式已经过期，不建议使用！"官方推荐我们在应用程序中使用磁场域和加速度传感器结合 SensorManager.getOrientation()来获得原始数据。下面通过一个实例来说明如何使用方向传感器。修改 MainActivity：

```
public class MainActivity extends AppCompatActivity implements SensorEventListener {

    private SensorManager sensorManager;
    private Sensor accelerometerSensor;
    private Sensor magneticFieldSensor;
    private float[] accelerometerValues = new float[3];
    private float[] magneticValues = new float[3];
    //旋转矩阵，用来保存磁场和加速度的数据
    private float[] r = new float[9];
    //模拟方向传感器的数据（原始数据为弧度）
```

```
    private float[] values = new float[3];

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        getSensorManager();
    }

    private void getSensorManager() {
        sensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
        accelerometerSensor = sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
        magneticFieldSensor = sensorManager.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD);
    }

    @Override
    protected void onResume() {
        super.onResume();
        //推荐在此解除注册
        if (sensorManager != null) {
            sensorManager.registerListener(this, accelerometerSensor,
                    SensorManager.SENSOR_DELAY_NORMAL);
            sensorManager.registerListener(this, magneticFieldSensor,
                    SensorManager.SENSOR_DELAY_NORMAL);
        }
    }

    @Override
    protected void onPause() {
        super.onPause();
        if (sensorManager != null) {
            //解除注册
            sensorManager.unregisterListener(this, accelerometerSensor);
            sensorManager.unregisterListener(this, magneticFieldSensor);
        }
    }

    @Override
    public void onSensorChanged(SensorEvent event) {
        if (event.sensor.getType() == Sensor.TYPE_ACCELEROMETER) {
            //这里是对象，需要克隆一份，否则共用一份数据
            accelerometerValues = event.values.clone();
        } else if (event.sensor.getType() == Sensor.TYPE_MAGNETIC_FIELD) {
            //这里是对象，需要克隆一份，否则共用一份数据
```

```
                magneticValues = event.values.clone();
            }
            /**
             * 填充旋转数组 r
             * r：要填充的旋转数组
             * I:将磁场数据转换进实际的重力坐标中，一般默认情况下可以设置为 null
             * gravity:加速度传感器数据
             * geomagnetic：地磁传感器数据
             */
            SensorManager.getRotationMatrix(r, null, accelerometerValues, magneticValues);
            /**
             * R：旋转数组
             * values ：模拟方向传感器的数据
             */
            SensorManager.getOrientation(r, values);

            float degree = (float) Math.toDegrees(values[0]);
            Log.i("指南针","当前手机角度为："+degree);
        }

        @Override
        public void onAccuracyChanged(Sensor sensor, int accuracy) {
            //注册的 Sensor 精度发送变化时，在此处处理
        }
    }
```

借助注释理解上述代码应该很容易。简单地说就是想要获取方向信息，需要使用加速度传感器和磁场域，然后分别获取它们的结果，然后使用 SensorManager.getRotationMatrix(r, null, accelerometerValues, magneticValues)给 r 数组赋值，再使用 SensorManager.getOrientation(r, values)从 r 数组中解析出方向信息，并赋值给 values 数组。values 数组中的第一个值就代表手机当前的角度。

需要注意的是，加速度传感器和磁场域在赋值的时候一定要调用 values 数组的 clone()方法，不然 accelerometerValues 和 magneticValues 将会指向同一个引用。另外，accelerometerValues 和 magneticValues 两个数组必须以成员变量的形式进行声明，而不能是局部变量。

运行程序，就可以在控制台看到手机指向的角度了，Log 如下：

```
15740-15740/com.buaa.sensor I/指南针：当前手机角度为：-131.16196
15740-15740/com.buaa.sensor I/指南针：当前手机角度为：-178.47978
15740-15740/com.buaa.sensor I/指南针：当前手机角度为：129.70839
15740-15740/com.buaa.sensor I/指南针：当前手机角度为：165.06802
15740-15740/com.buaa.sensor I/指南针：当前手机角度为：128.89392
15740-15740/com.buaa.sensor I/指南针：当前手机角度为：137.37225
15740-15740/com.buaa.sensor I/指南针：当前手机角度为：-120.013145
15740-15740/com.buaa.sensor I/指南针：当前手机角度为：-162.30382
15740-15740/com.buaa.sensor I/指南针：当前手机角度为：-139.26479
```

需要说明的是，这里的角度取值范围是－180°～180°，90°代表东方，－90°代表西方，±180°代表南方，0 度代表北方。

关于传感器的内容就讲到这里。就像前文所说的，Android 中传感器的种类很多，这里讲解的只是其中的一小部分，但是用法大致相同，读者如果想要学习更多关于传感器的知识，可以通过阅读文档、参考本节内容来进行学习。

12.2　地理位置定位

现在大家即使没有使用过滴滴打车，也应该使用过饿了么或者美团外卖。这些应用能够准确定位，使用的是地理信息技术，或者用一个比较火的词 LBS（基于位置的服务）来形容。随着移动互联网大潮的到来以及滴滴、饿了么、美团、优步这样的企业越来越多，地理信息技术也开始慢慢升温。本节将讲解在 Android 中如何使用地理信息技术来实现对地理位置的定位。

12.2.1　LocationManager 的使用

在 Android 中进行地理位置定位主要使用的类是 LocationManager。使用 LocationManager 的方法类似于使用其他服务，只需要通过调用 getSystemService()方法就可以实例化该类的对象，并获得它的引用。当然这里需要传入的参数是“Context.LOCATION_SERVICE”。当获取到 LocationManager 对象之后，直接通过 LocationManager 调用 getLastKnownLocation()方法就可以获得 Location 类对象，而 Location 类正是保存着位置信息的类。

getLastKnownLocation()方法需要传入位置提供者来确定设备当前的位置。Android 中常用的位置提供者有 LocationManager.GPS_PROVIDER 与 LocationManager.NETWORK_PROVIDER，分别指使用 GPS 和网络进行定位。其中，GPS 定位的精准度比较高，但是非常耗电，而网络定位的精准度稍差，耗电量比较少。在开发时，应该根据自己的实际情况来选择使用哪一种位置提供器，当位置精度要求非常高的时候最好使用 GPS_PROVIDER，在一般情况下使用 NETWORK_PROVIDER 会更好。

同时，如果需要检测位置变化情况，可以使用 locationManager 调用 requestLocationUpdates()来注册 LocationListener。

另外，Location 所包含的位置信息是经纬度信息，如果想要获取具体的地址信息，可以使用 Geocoder 类进行地理位置解析。该类用于获取地理位置的前向编码和反向编码，前向编码是根据地址获取经纬度；反向编码是根据经纬度获取对应的详细地址。Geocoder 请求的是一个后台服务，但是该服务不包括在标准 Android framework 中。因此如果当前设备不包含 location services，则 Geocoder 返回的地址或者经纬度为空。当然你可以使用 Geocoder 的 isPresent()方法来判断当前设备是否包含地理位置服务。而且由于国内使用不了 Google Services 服务，因此一般的手机厂商都会在自己的手机内内置百度地图服务或者高德地图服务来替代 Google Services 服务。

下面通过一个实例来进行说明。创建一个新的项目，直接修改 MainActivity：

```
public class MainActivity extends AppCompatActivity implements LocationListener {
    private LocationManager locationManager;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
    @Override
    protected void onResume() {
        super.onResume();
        initLocation();
    }
    @Override
    protected void onDestroy() {
        super.onDestroy();
        if (locationManager != null) {
            checkPermission(new String[]{
                    Manifest.permission.ACCESS_COARSE_LOCATION,
                    Manifest.permission.ACCESS_FINE_LOCATION
            });
            //解除监听
            locationManager.removeUpdates(this);
            locationManager = null;
        }
    }

    private void initLocation() {
        checkPermission(new String[]{
                Manifest.permission.ACCESS_COARSE_LOCATION,
                Manifest.permission.ACCESS_FINE_LOCATION
        });
        locationManager = (LocationManager) getSystemService(Context.
                LOCATION_SERVICE);
        //这里使用 GPS 位置提供者作为案例
        Location location = locationManager.getLastKnownLocation(LocationManager.GPS_PROVIDER);
        //监听位置的变化，每隔两秒且距离差距为 10 米时更新位置信息，这助于控制电量
        locationManager.requestLocationUpdates(LocationManager.GPS_PROVIDER, 2000, 10, this);
        if (location != null) {
            Log.i("location", "纬度： " + location.getLatitude() + ",经度" + location.getLongitude());
            getLocation(location);
        }
    }
```

```
private void getLocation(Location location) {
    Geocoder geocoder = new Geocoder(this);
    try {
        //使用 geocoder 获取具体的地址，参数为纬度和经度
        List<Address> addresses = geocoder.getFromLocation(
                location.getLatitude(), location.getLongitude(), 1);
        Address address = addresses.get(0);
        Log.i("location", address.getAddressLine(0) +
                address.getAddressLine(1) + address.getFeatureName());
    } catch (IOException e) {
        e.printStackTrace();
    }
}

@Override
public void onLocationChanged(Location location) {
    //当符合监听条件时，会更新地理位置
    Log.i("location", "纬度：" +
            location.getLatitude() + ",经度" + location.getLongitude());
    getLocation(location);
}
@Override
public void onStatusChanged(String provider, int status, Bundle extras) {
}
@Override
public void onProviderEnabled(String provider) {
}
@Override
public void onProviderDisabled(String provider) {
}

private void checkPermission(String[] permissions) {
    int permission_granted = PackageManager.PERMISSION_GRANTED;
    boolean flag = false;
    for (int i = 0; i < permissions.length; i++) {
        int checkPermission = ActivityCompat.checkSelfPermission
                (this, permissions[i]);
        if (permission_granted != checkPermission) {
            flag = true;
            break;
        }
    }
```

```
            if (flag) {
                ActivityCompat.requestPermissions(this, permissions, 111);
                return;
            }
        }

        @Override
        public void onRequestPermissionsResult(
                int requestCode, String[] permissions, int[] grantResults) {
            switch (requestCode) {
                case 111:
                    if (grantResults[0] == PackageManager.PERMISSION_GRANTED) {
                        Toast.makeText(this, "获取权限成功",
                                Toast.LENGTH_SHORT).show();
                    } else {
                        Toast.makeText(this, "获取权限失败",
                                Toast.LENGTH_SHORT).show();
                    }
                    break;
                default:
                    super.onRequestPermissionsResult(requestCode, permissions, grantResults);
            }
        }
    }
```

结合注释和前文所描述的使用 LocationManager 的步骤，应该能够很容易理解代码的内容。这里需要注意的就是使用地理位置信息属于危险权限，需要进行动态权限的检查。除此之外，还需要在 AndroidManifest.xml 中显式加入对权限的声明：

```
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
```

打开 GPS，然后运行应用，在控制台上就会打印出经纬度信息以及地址信息了。此时移动手机，位置信息还会进行更新。Log 信息如下：

```
22630-22630/com.buaa.lbs I/location: 纬度：31.77904876,经度117.18329526
22630-22630/com.buaa.lbs I/location: 中国安徽省合肥市蜀山区芙蓉社区容成路48号馨苑小区（容成路）
22630-22630/com.buaa.lbs I/location: 纬度：31.77904876,经度117.18329526
22630-22630/com.buaa.lbs I/location: 中国安徽省合肥市蜀山区芙蓉社区容成路48号馨苑小区（容成路）
22630-22630/com.buaa.lbs I/location: 纬度：31.77904284,经度117.1833039
22630-22630/com.buaa.lbs I/location: 中国安徽省合肥市蜀山区芙蓉社区容成路48号馨苑小区（容成路）
22630-22630/com.buaa.lbs I/location: 纬度：31.77903263,经度117.18331223
22630-22630/com.buaa.lbs I/location: 中国安徽省合肥市蜀山区芙蓉社区容成路48号馨苑小区（容成路）
```

本例中使用 GPS 的方式获取地理位置信息，使用网络的话，开发方式是一样的，只不过要额外加上网络的权限。读者可以自行尝试开发。

12.2.2 使用高德地图

相信使用过优步或者滴滴的读者都知道它们使用的是第三方的地图，即百度地图和高德地图。和它们一样，在我们的应用中也可以加入地图功能。在地图领域，相对领先的是百度地图和谷歌地图，次之则是高德地图。但是由于谷歌在某些方面的原因，在国内无法提供服务，虽然相较于高德地图，百度地图的地图服务更加准确、UI 更加友好，但是百度地图的开发更加适合 IDE 为 Eclipse 的开发者。而把 Android Studio 作为 IDE 的我们，高德地图是更好的选择。而且高德地图看起来是未来的趋势，现在很多 LBS 应用也开始渐渐地使用高德地图。因此这里我们选择使用高德地图来完成应用。

1. 申请 API Key

要想使用高德地图 SDK，就必须申请一个高德地图的 API Key。而想要申请 API Key 就必须先成为高德地图开放平台开发者。注册地图开放平台开发者的链接为“https://id.amap.com/register/index?ref=http%3A%2F%2Flbs.amap.com%2F”，进入此页面，填写相关信息完成注册，然后点击“成为开发者”，然后填写个人信息并进行邮箱验证之后即可成为高德地图开放平台开发者！这一系列的注册问都很简单，这里不详细介绍。

在刚刚注册完成的页面中会有“申请 KEY”的选项，选择它之后会进入控制台（或者之后登录高德开放平台，直接进入控制台），如图 12-2 所示。



图 12-2 高德开发平台的控制台界面

这时如果已有应用就点击“添加新 Key”，否则点击“创建新应用”来创建一个新的应用。创建新应用很简单，只需要输入应用名和应用类型即可，而添加新 Key 则相对较复杂。点击“添加新 Key”按钮后需要填写若干信息，如图 12-3 所示。

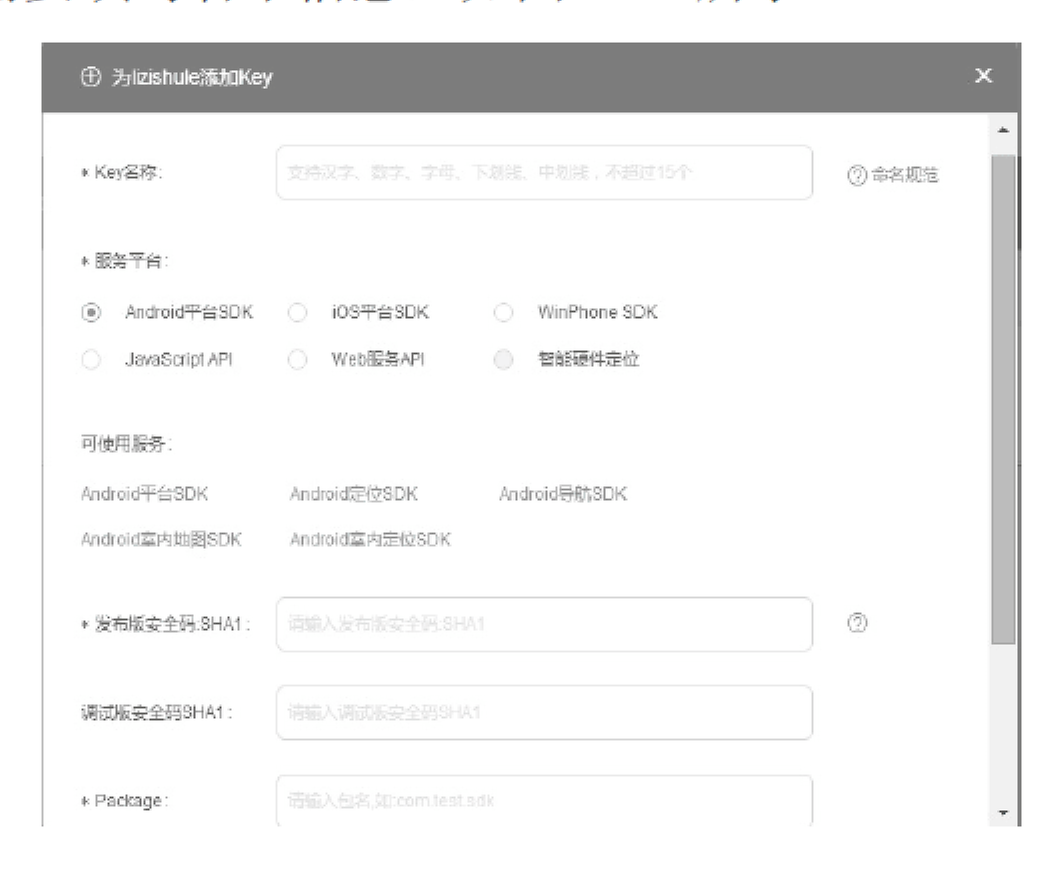


图 12-3 添加新的 API Key

这里 Key 名称只要不违反规则，叫什么都可以。服务平台自然选择 Android 平台 SDK。Package 填入应用包名，即 AndroidManifest.xml 中的包名。除此之外，还需要填写 SHA1，这个相对比较复杂。在 Android Studio 中获取 SHA1，可以打开 Android Studio 的 Terminal 工具，输入命令“keytool -v -list -keystore keystore 文件路径”，然后输入 Keystore 密码。

但是，从步骤来看获取 SHA1 还需要 Keystore。读者可能比较迷惑，这里的 Keystore 是什么、怎么获取它呢？Android 独有的安全机制除了权限机制外，还有签名机制，而 Keystore 就是使用特殊的 Key 来给 Android 应用做签名。签名机制主要用在以下两个主要场合：升级 App 和权限检查。这样可以防止已安装的应用被恶意的第三方覆盖或替换掉。Android 系统要求只有签名的 APK 文件才可以安装。我们在之前的学习中使用 Android Studio 开发出一个应用之后，安装到手机上也是需要进行签名的，只不过使用的是默认的 KeyStore，所以读者并没有感知到。默认的 Keystore 是 Android 为了方便调试而提供的，位置在“C:\Users\Administrator\.android”下，如图 12-4 所示。

avd	2016/9/5 4:57	文件夹	
breakpad	2016/7/13 11:14	文件夹	
cache	2016/7/13 16:26	文件夹	
monitor-workspace	2016/7/13 11:15	文件夹	
studio	2016/6/3 17:45	文件夹	
.emu-update-last-check	2016/9/14 16:37	EMU-UPDATE-L...	1 KB
adbkey	2016/5/29 13:42	文件	2 KB
adbkey.pub	2016/5/29 13:42	PUB 文件	1 KB
androidtool.cfg	2016/8/13 21:19	CFG 文件	1 KB
androidwin.cfg	2016/8/13 21:19	CFG 文件	1 KB
ddms.cfg	2016/9/10 13:45	CFG 文件	1 KB
debug.keystore	2016/6/3 18:01	KEYSTORE 文件	3 KB
default.keyset	2016/7/13 11:14	KEYSET 文件	1 KB
modem-nv-ram-5554	2016/9/14 16:37	文件	1 KB
modem-nv-ram-5556	2016/9/5 5:12	文件	1 KB
repositories.cfg	2016/8/13 21:19	CFG 文件	1 KB
sites-settings.cfg	2016/7/13 16:27	CFG 文件	2 KB

图 12-4　Android 默认的 Keystore

在 Android 应用正式发布到应用市场的时候，还需要新建一个发布版的 Keystore，这将在最后一章介绍，这里就使用 debug 版的 Keystore 来获取 SHA1 值。这里需要 Keystore 的密码，debug.store 的密码为“android”。此时按照前述步骤就可以获取到 SHA1 值了，如图 12-5 所示。

```
Microsoft Windows [版本 6.1.7601]
版权所有 (c) 2009 Microsoft Corporation。保留所有权利。

E:\Autonavi\SDKDemo\androidpublishdemo\AMap_Location_API_Demo_V2.x\android_studi
o\AMapLocationDemo>C:\Java\jdk1.6.0_24\bin\keytool -list -v -keystore D:\keystore\debug.keystore
输入keystore密码:

Keystore 类型: JKS
Keystore 提供者: SUN

您的 keystore 包含 1 输入

别名名称: androiddebugkey
创建日期: 2014-9-18
项类型: PrivateKeyEntry
认证链长度: 1
认证 [1]:
所有者:CN=Android Debug, O=Android, C=US
签发人:CN=Android Debug, O=Android, C=US
序列号:541ab9ee
有效期: Thu Sep 18 18:54:38 CST 2014 至Sat Sep 10 18:54:38 CST 2044
证书指纹:
         MD5:16:BE:38:29:93:7D:92:6F:F1:11:06:C1:7E:0F:1D:CB
         SHA1:5E:F3:0F:CF:86:BA:74:96:E7:C3:F2:BC:24:CD:38:B1:4A:F2:A2:63
         签名算法名称:SHA1withRSA
```

JDK安装路径
keystore文件路径
在此处输入keystore的密码
keystore中key的别名，用于在Android Studio中配置签名文件使用
keystore对应的SHA1，在控制台申请Key时使用

图 12-5　根据 Keystore 生成申请高德 API Key 所需的 SHA1 值

将 SHA1 值和其他选项都填好后，点击“提交”就会生成一个 Key 值。接下来我们就可以获取 SDK 进行地图开发了。

2. 获取高德地图 SDK 工具包

进入高德地图开发者平台的首页（http://lbs.amap.com/），可以看到在 Android 平台上，它提供的功能有很多，比如地图功能、定位功能、导航功能、室内定位、室内地图等。本例中我们以地图功能为例来进行讲解。要实现地图功能需要下载地图 SDK，点击图 12-6 中的“地图 SDK”选项，会进入开发文档的界面。此时点击左侧边栏下方的“相关下载”就可以进入地图 SDK 的下载页面了。高德提供了两种不同的下载方式，以供开发者选择，如图 12-7 所示。

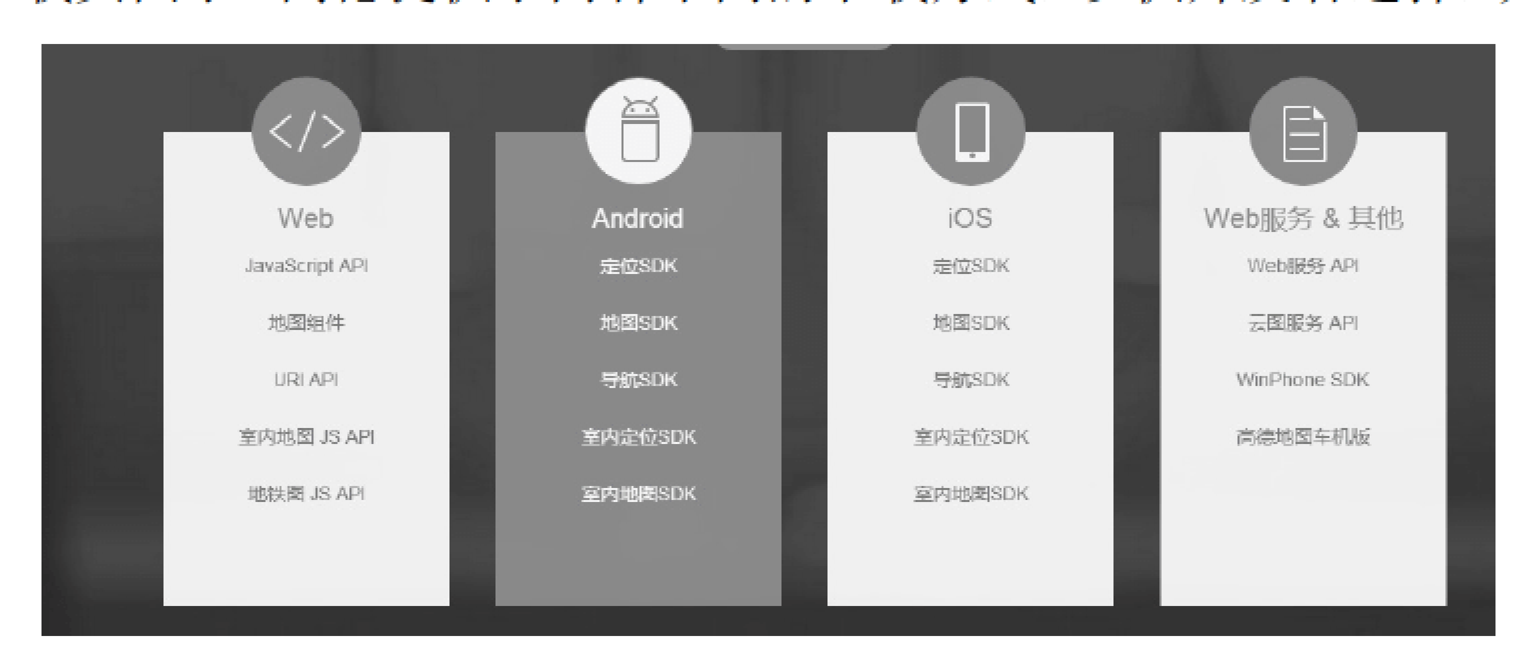


图 12-6　选择 Android 中的“地图 SDK”



图 12-7　下载所需要的 SDK

下载完成之后，将 jar 包加入 libs 文件夹（以 Project 形式打开项目时，在 app 模块下会显示 libs 文件夹）下就可以了，如图 12-8 所示。

libs
AMap_2DMap_V2.9.0_20160525.jar
AMap_Location_V2.4.1_20160414.jar
AMap_Search_V3.4.0_20160811.jar
Volley.jar

图 12-8　将 SDK 中的 jar 文件加入项目中

3. 使用高德地图 SDK 进行开发

直接修改 activity_main.xml 文件：

```
<?xml version="1.0" encoding="utf-8"?>
<com.amap.api.maps2d.MapView
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/map"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

这里的 com.amap.api.maps2d.MapView 控件是高德地图提供的专门用于展示地图的控件。修改 MainActivity：

```
public class MainActivity extends AppCompatActivity {
    private MapView mapView;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        mapView = (MapView) findViewById(R.id.map);
        mapView.onCreate(savedInstanceState);
    }
    @Override
    protected void onResume() {
        super.onResume();
        mapView.onResume();
    }
    @Override
    protected void onPause() {
        super.onPause();
        mapView.onPause();
    }
    @Override
    protected void onSaveInstanceState(Bundle outState) {
        super.onSaveInstanceState(outState);
        mapView.onSaveInstanceState(outState);
    }
    @Override
    protected void onDestroy() {
        super.onDestroy();
        mapView.onDestroy();
    }
}
```

代码中的操作很简单，首先通过 findViewById()方法获取 MapView 对象，然后通过 MapView 对象在 Activity 的各个声明周期方法中调用对应的方法。通过这么简单的操作，就可以在应用中显示地图了。虽然在 Activity 中并没有使用需要申请权限的内容，但是使用的 jar 文件中有所涉及，所以还需要在 AndroidManifest.xml 中进行权限声明，而使用高德地图所需要的 Api Key 也需要在 AndroidManifest.xml 中配置，所以修改 AndroidManifest.xml 文件如下：

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.buaa.map">
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <meta-data
            android:name="com.amap.api.v2.apikey"
            android:value="这里填入你所申请的 api key"></meta-data>
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

完成之后，运行程序，展示地图，如图 12-9 所示。



图 12-9　使用高德展示地图控件来展示地图

高德地图默认显示的是祖国的首都北京。打开地图的时候，用户可能希望首先展示在面前的是他所处的位置，所以修改代码，使得应用可以直接定位到当前位置，并展示地图：

```
public class MainActivity extends AppCompatActivity implements LocationSource,
        AMapLocationListener {
    private MapView mapView;
    private AMap aMap;
    private OnLocationChangedListener listener;
    private AMapLocationClient locationClient;
    private AMapLocationClientOption locationOption;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        mapView = (MapView) findViewById(R.id.map);
        mapView.onCreate(savedInstanceState);
        init();
    }

    private void init() {
        if (aMap == null) {
            aMap = mapView.getMap();
            setUpMap();
        }
    }

    // 设置一些 amap 的属性
    private void setUpMap() {
        // 自定义系统定位小蓝点
        MyLocationStyle myLocationStyle = new MyLocationStyle();
        // 设置小蓝点的图标
        myLocationStyle.myLocationIcon(BitmapDescriptorFactory
                .fromResource(R.drawable.location_marker));
        // 设置圆形的边框颜色
        myLocationStyle.strokeColor(Color.BLACK);
        // 设置圆形的填充颜色
        myLocationStyle.radiusFillColor(Color.argb(100, 0, 0, 180));
        // 设置圆形的边框粗细
        myLocationStyle.strokeWidth(1.0f);
        aMap.setMyLocationStyle(myLocationStyle);
        // 设置定位监听
        aMap.setLocationSource(this);
```

```
        // 设置默认定位按钮是否显示
        aMap.getUiSettings().setMyLocationButtonEnabled(true);
        // 设置为 true 表示显示定位层并可触发定位，false 表示隐藏定位层并不可触发定位，默认是 false
        aMap.setMyLocationEnabled(true);
    }

    @Override
    protected void onResume() {
        super.onResume();
        mapView.onResume();
    }
    @Override
    protected void onPause() {
        super.onPause();
        mapView.onPause();
        deactivate();
    }
    @Override
    protected void onSaveInstanceState(Bundle outState) {
        super.onSaveInstanceState(outState);
        mapView.onSaveInstanceState(outState);
    }
    @Override
    protected void onDestroy() {
        super.onDestroy();
        mapView.onDestroy();
    }

    @Override
    public void onLocationChanged(AMapLocation amapLocation) {
        if (listener != null && amapLocation != null) {
            if (amapLocation != null
                    && amapLocation.getErrorCode() == 0) {
                listener.onLocationChanged(amapLocation);// 显示系统小蓝点
            } else {
                String errText = "定位失败," + amapLocation.getErrorCode() + ": " +
amapLocation.getErrorInfo();
                Log.e("AmapErr", errText);
            }
        }
    }
```

```
    //激活定位
    @Override
    public void activate(OnLocationChangedListener listener) {
        this.listener = listener;
        if (locationClient == null) {
            locationClient = new AMapLocationClient(this);
            locationOption = new AMapLocationClientOption();
            //设置定位监听
            locationClient.setLocationListener(this);
            //设置为高精度定位模式
            locationOption.setLocationMode
                    (AMapLocationClientOption.AMapLocationMode.Hight_Accuracy);
            //设置定位参数
            locationClient.setLocationOption(locationOption);
            /**
             * 此方法为每隔固定时间会发起一次定位请求，为了减少电量消耗或网络流量消耗，
             * 注意设置合适的定位时间间隔（支持最小间隔为 2000ms）
             * 定位结束后，在合适的生命周期调用 onDestroy()方法
             * 在单次定位情况下，定位无论成功与否，都无须调用 stopLocation()方法移除请求，
             *定位 sdk 内部会移除
             */
            locationClient.startLocation();
        }
    }

    // 停止定位
    @Override
    public void deactivate() {
        listener = null;
        if (locationClient != null) {
            locationClient.stopLocation();
            locationClient.onDestroy();
        }
        locationClient = null;
    }
}
```

与前一个只展示地图的实例相比，这里使用定位功能，然后根据定位展示到用户所在地方。结合注释和前一个实例应该很容易理解这里的逻辑。要想实现定位，还需要修改 AndroidManifest.xml 增加权限并配置一个用来定位的 Service，代码如下：

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.buaa.map">

    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
    <uses-permission android:name="android.permission.READ_PHONE_STATE" />
    <uses-permission android:name="android.permission.CHANGE_WIFI_STATE" />
    <uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
    <uses-permission android:name="android.permission.CHANGE_CONFIGURATION" />
    <uses-permission android:name="android.permission.WAKE_LOCK" />
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <meta-data
            android:name="com.amap.api.v2.apikey"
            android:value="0b30fb31c0abf82c2a95aa79e42dc85f"></meta-data>

        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <!-- 定位需要的服务 使用 2.0 的定位需要加上这个 -->
        <service android:name="com.amap.api.location.APSService"></service>
    </application>
</manifest>
```

完成之后，运行程序，就可以看到地图已经定位用户所在地了，如图 12-10 所示。

高德地图中的功能很多，这里讲到的只是高德地图中很小的一部分功能。比如公交线路的查询、根据输入查询位置，这些都是常用的功能。如果读者有兴趣，可以在下载 SDK 时下载 demo 进行学习。

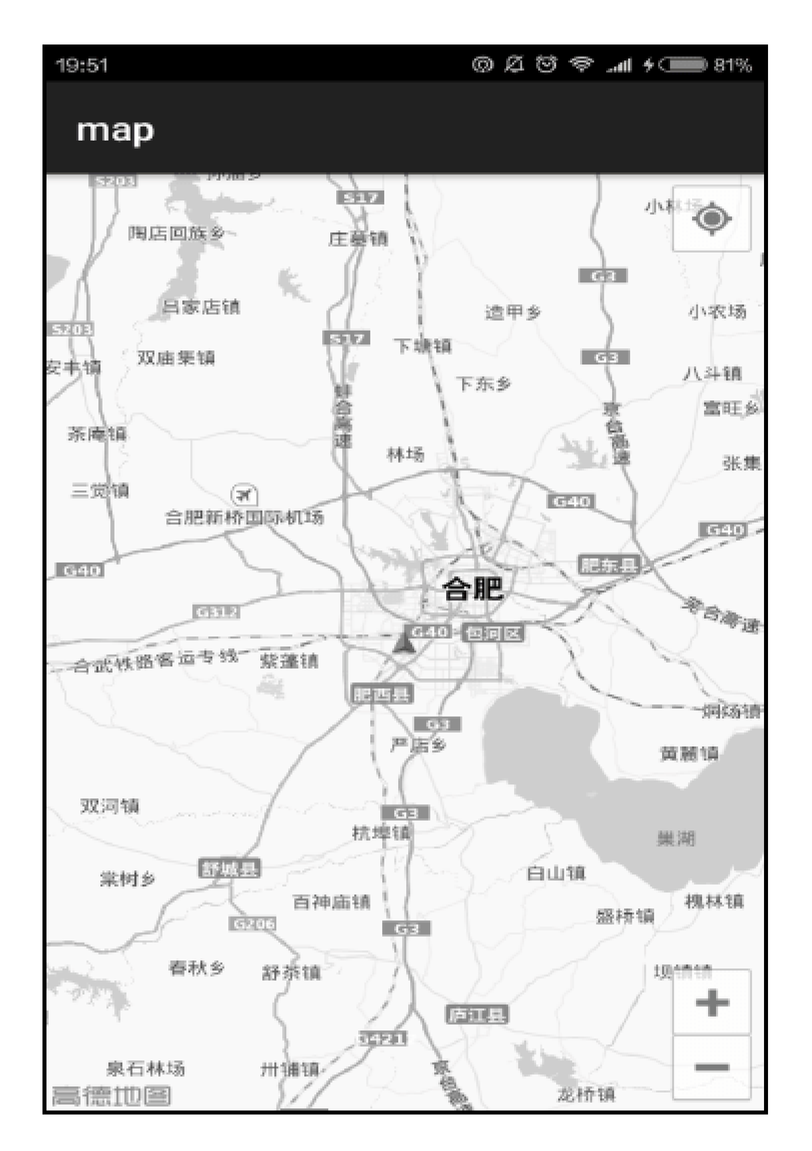


图 12-10　展示所在城市地图

12.3　小　　结

本章主要以传感器和地理信息技术为例讲解了 Android 中具有特色的一些功能。在传感器中，我们介绍了加速度传感器、光照传感器、方向传感器的使用，并根据它们的原理开发了具有特殊功能的小应用。在地理信息技术中，我们讲解了如何进行定位，以及如何使用 Geocoder 类进行地理位置解析来获取具体的位置。同时我们还讲解了如何使用第三方工具高德地图来展示位置。

不管是传感器还是地理信息技术，抑或是高德地图，如果想要详细介绍它们，甚至可以各用一本书来讲解。本章都只是帮助读者入门，如果读者想要深入挖掘这几种技术，可以通过阅读文档或者官方 demo 来进行学习。



第 13 章 VR 开发入门

2014 年 3 月 26 日，美国社交网络平台 Facebook 宣布斥资 20 亿美元收购沉浸式虚拟现实技术公司 Oculus VR；在 2015 年 3 月的 MWC2015 上，HTC 与曾制作 Portal 和 Half-Life 等独创游戏的 Valve 联合开发的虚拟现实（VR）头盔 HTC Vive 正式亮相……这一系列事件的发生，标志着 VR 正在逐步进入爆发期，一个字——“火”。

对于非 IT 圈的人来说，VR 到底是什么、VR 的前景到底如何，他们并不了解。而对于 Android 开发者来说，这又意味着什么，Android 开发者能够开发 VR 应用吗？这些就是本章将要讲解的内容。

13.1 详解 VR

对于非 IT 圈的人来说，VR 是那么神秘。本节将从 VR 是什么、VR 的关键技术、VR 发展历程、VR 面临的技术瓶颈、VR 的市场前景等多个方面来详细阐述 VR。

13.1.1 VR 是什么

VR 是 Virtual Reality 的缩写，中文的意思是虚拟现实。这个概念是在 20 世纪 80 年代初提出来的，具体是指借助计算机及最新传感器技术创造的一种崭新的人机交互手段。

虚拟现实技术是仿真技术的一个重要方向，是仿真技术与计算机图形学人机接口技术、多媒体技术、传感技术、网络技术等多种技术的集合，是一门富有挑战性的交叉技术前沿学科和研究领域。虚拟现实技术（VR）主要包括模拟环境、感知、自然技能和传感设备等方面。模拟环境是由计算机生成的、实时动态的三维立体逼真图像。感知是指理想的 VR 应该具有一切人所具有的感知。除计算机图形技术所生成的视觉感知外，还有听觉、触觉、力觉、运动等感知，甚至包括嗅觉和味觉等，也称为多感知。自然技能是指人的头部转动，眼睛、手势或其他人体行为动作，由计算机来处理、与参与者的动作相适应的数据，并对用户的输入做出实时响应，分别反馈到用户的五官。传感设备是指三维交互设备。

VR 包括如下特征：

- 多感知性：指除一般计算机所具有的视觉感知外，还有听觉感知、触觉感知、运动感知，甚至还包括味觉、嗅觉、感知等。理想的虚拟现实应该具有一切人所具有的感知功能。
- 存在感：指用户感到作为主角存在于模拟环境中的真实程度。理想的模拟环境应该达到使用户难辨真假的程度。
- 交互性：指用户对模拟环境内物体的可操作程度和从环境得到反馈的自然程度。
- 自主性：指虚拟环境中的物体依据现实世界物理运动定律动作的程度。

13.1.2 VR 的关键技术

虚拟现实是多种技术的综合，包括实时三维计算机图形技术，广角（宽视野）立体显示技术，对观察者头、眼和手的跟踪技术，以及触觉/力觉反馈、立体声、网络传输、语音输入输出技术等。下面分别对这些技术加以说明。

1. 实时三维计算机图形

相比较而言，利用计算机模型产生图形图像并不是太难的事情。如果有足够准确的模型，又有足够的时间，我们就可以生成不同光照条件下各种物体的精确图像，但是这里的关键是实时。例如在飞行模拟系统中图像的刷新相当重要，同时对图像质量的要求也很高，再加上非常复杂的虚拟环境，问题就变得相当困难。

2. 显示

人看周围的世界时，由于两只眼睛的位置不同，得到的图像略有不同，这些图像在脑子里融合起来就形成了一个关于周围世界的整体景象，这个景象中包括了距离远近的信息。当然，距离信息也可以通过其他方法获得，例如眼睛焦距的远近、物体大小的比较等。

在 VR 系统中，双目立体视觉起了很大作用。用户的两只眼睛看到的不同图像是分别产生的，显示在不同的显示器上。有的系统采用单个显示器，但用户带上特殊的眼镜后，一只眼睛只能看到奇数帧图像，另一只眼睛只能看到偶数帧图像，奇、偶帧之间的不同（视差）就产生了立体感。

用户（头、眼）的跟踪：在人造环境中，每个物体相对于系统的坐标系都有一个位置与姿态，而用户也是如此。用户看到的景象是由用户的位置和头（眼）的方向来确定的。

跟踪头部运动的虚拟现实头套：在传统的计算机图形技术中，视场的改变是通过鼠标或键盘来实现的，用户的视觉系统和运动感知系统是分离的，而利用头部跟踪来改变图像的视角，用户的视觉系统和运动感知系统之间就可以联系起来，感觉更逼真。另一个优点是，用户不仅可以通过双目立体视觉去认识环境，还可以通过头部的运动去观察环境。

在用户与计算机的交互中，键盘和鼠标是目前最常用的工具，但对于三维空间来说，它们都不太适合。在三维空间中因为有 6 个自由度，我们很难找出比较直观的办法把鼠标的平面运动映射成三维空间的任意运动。现在，已经有一些设备可以提供 6 个自由度，如 3Space 数字化仪和 SpaceBall 空间球等。另外，一些性能比较优异的设备是数据手套和数据衣。

3. 声音

人能够很好地判定声源的方向。在水平方向上，我们靠声音的相位差及强度的差别来确定声音的方向，因为声音到达两只耳朵的时间或距离有所不同。常见的立体声效果就是靠左右耳听到在不同位置录制的不同声音来实现的，所以会有一种方向感。在现实生活里，当头部转动时，听到的声音的方向就会改变。但目前在 VR 系统中，声音的方向与用户头部的运动无关。

4. 感觉反馈

在一个 VR 系统中，用户可以看到一个虚拟的杯子。你可以设法去抓住它，但是你的手没有真正接触杯子的感觉，并有可能穿过虚拟杯子的“表面”，而这在现实生活中是不可能的。解决这一问题的常用装置是在手套内层安装一些可以振动的触点来模拟触觉。

5. 语音

在 VR 系统中，语音的输入输出也很重要。这就要求虚拟环境能听懂人的语言，并能与人实时交互。而让计算机识别人的语音是相当困难的，因为语音信号和自然语言信号有其“多边性”和复杂性。例如，连续语音中词与词之间没有明显的停顿，同一词、同一字的发音受前后词、字的影响，不仅不同人说同一词会有所不同，就是同一人发音也会受到心理、生理和环境的影响而有所不同。

使用人的自然语言作为计算机输入目前有两个问题，首先是效率问题，为便于计算机理解，输入的语音可能会相当啰嗦。其次是正确性问题，计算机理解语音的方法是对比匹配，而没有人的智能。

13.1.3 VR 发展历程

VR 经历了 3 次热潮：第一次源于 20 世纪 60 年代，确立了 VR 技术原理；第二次发生在 20 世纪 90 年代，VR 试图商业化但未能成功；目前正处于第三次热潮前期，以 Facebook 20 亿美元收购 Oculus 为标志，全球范围内掀起了 VR 商业化、普及化的浪潮。

第一次热潮发生在 20 世纪 60 年代，科学家们建立了 VR 的基础原理和产品光学构造。20 世纪 60 年代，电影摄影师 Morton Heilig 提交了一款 VR 设备的专利申请文件，专利文件上的描述是“用于个人使用的立体电视设备”。尽管这款设计来自于 50 多年前，但可以看出与 Oculus Rift、Google Cardboard 有很多相似之处。1967 年，Heilig 又构造了一个多感知仿环境的虚拟现实系统 Sensorama Simulator，这也是历史上第一套 VR 系统，它能够提供真实的 3D 体验，例如用户在观看摩托车形式的画面时，不仅能看到立体、彩色、变化的街道画面，还能听到立体声，感受到行车的颠簸、扑面而来的风以及闻到花的芳香。1968 年美国计算机图形学之父 Ivan Sutherlan 在哈佛大学组织开发了第一个计算机图形驱动的头盔显示器 HMD 及头部位置跟踪系统，是 VR 发展史上一个重要的里程碑。进入 20 世纪 80 年代，VR 相关技术在飞行、航天等领域得到比较广泛的应用。

第二次热潮发生在 20 世纪 90 年代，这是一次如火如荼的商业化热潮，但最终没能获得成功。1989 年 Jaron Lanier 首次提出 Virtual Reality 的概念，被称为“虚拟现实之父”。1991 年，一款名为“Virtuality 1000CS”的设备出现在消费市场中，笨重的外形、单一的功能和昂贵的价格使其并未得到消费者的认可，但掀起了一个 VR 商业化的浪潮，世嘉、索尼、任天堂等都陆续推出了自己的 VR 游戏机产品。在这一轮商业化热潮中，由于光学、计算机、图形、数据等领域技术尚处于高速发展早期、产业链也不完备，并未得到消费者的积极响应。但此后，企业的 VR 商业化尝试一直没有停止。

第三次热潮源于 2014 年 Facebook 20 亿美元收购 Oculus，VR 商业化进程在全球范围内得到加速。2014 年 3 月 26 日，Oculus VR 被 Facebook 以 20 亿美元收购，再次引爆全球 VR 市场。三星、HTC、索尼、雷蛇、佳能等科技巨头组团加入，让人们看到这个行业正在蓬勃发展；国内，目前已经出现数百家 VR 领域创业公司，覆盖全产业链环节，例如交互、摄像、现实设备、游戏、视频等。暴风科技登录创业板，成为“虚拟现实第一股”，吸引更多创业者和投资者进入 VR 领域。

13.1.4 VR 在技术层面上的现状

到目录为止，VR 的发展还面临如下一些技术上的瓶颈。

- 硬件瓶颈。AR 对计算能力的要求比 VR 高一个数量级，目前的 CPU、GPU 无法支持，更无法保证在轻便的硬件上实现足够的计算速度、存储空间、传输速率和续航能力。
- 图像技术瓶颈。图像识别技术不成熟，特别是在复杂图形、动态图像、特殊场景（如夜间）等方面，信息筛选、识别的正确率和精确率均较低，远不足以支撑一款消费级产品；

实时三维建模技术缺乏：需要以图像识别技术作为基础，仅处于实验室阶段；精确定位技术误差大，远未到商用阶段。

- 数据瓶颈。在现实环境中实现无差别图像视频识别需要极其庞大的数据规模，如一条街道上，需要街景、人脸、服装等各种数据；目前数据的采集、存储、传输、分析技术都有需要解决的难题，仅海量数据的清洗、录入本身就是浩瀚的工程。

虽然目前 VR 产品的体验仍有很多局限，还不足以进入消费市场，但投资机构普遍重视、企业研发极其活跃，已经完成从无到有的冷启动。

VR 技术包括 4 项关键指标，领先厂商已经达标，VR 技术趋于成熟。这 4 项指标分别为：屏幕刷新率、屏幕分辨率、延迟和设备计算能力。目前高通骁龙 820 已经上市，19.3ms 内的延迟已经可以达到；90Hz 和 2K 屏幕已进入市场，可以提供基础级 VR 产品体验。同时，其他方面的技术（如输入设备在姿态矫正、复位功能、精准度、延迟等方面）持续改善；传输设备提速并无线化；更小体积硬件下的续航能力和存储容量不断提升；配套系统和中间件开发完善。

首先，VR 系统越发成熟。其实，目前 Windows、Android 系统已经能够较好地支持 VR 的软硬件、提供较好的体验，支撑消费级应用，而 Google、Oculus、Razer 还都在开发 VR 专用系统。

其次，核心技术将于明年普及。明年将有更多厂商和设备能够在核心技术参数上达到 VR 级，这是硬件和应用在消费市场爆发的必要条件。

再次，世界主流的 VR 硬件都将推出消费者版本。到目前为止，全球体验最好的 VR 硬件，包括 Oculus Rift、三星 Gear VR、Value&HTC Vive 和索尼 PlayStation VR，都仅推出了开发者版本，而这四大产品都将推出消费者版，这将直接引爆消费市场和应用开发者群体。

13.1.5 VR 当前市场现状

当前，VR 技术得到了业界的普遍关注，但这并不是首次。早在 20 世纪 90 年代就已经有 3D 游戏上市，VR 在当时也引发了类似于当前的关注度。例如，游戏方面有 Virtuality 的 VR 游戏系统和任天堂的 Vortual Boy 游戏机，电影方面有《异度空间》（Lawnmower Man）、《时空悍将》（Virtuosity）和《捍卫机密》（Johnny Mnemonic），书籍方面有《雪崩》（Snow Crash）和《桃色机密》（Disclosure）。但是，当时的 VR 技术没有跟上媒体不切合实际的想象。例如，3D 游戏画质较差、价格高、时间延迟、设备计算能力不足等。最终，这些产品以失败告终，因为消费者对这些技术并不满意，所以第一次 VR 热潮就此消退。

到了 2014 年，Facebook 以 20 亿美元收购 Oculus 后，类似的 VR 热再次袭来。在过去的两年中，VR/AR 领域共进行了 225 笔风险投资，投资额达到了 35 亿美元。与 20 世纪 90 年代的失败相比，当前的 VR 热有什么不同呢？答案在于技术。当前计算机的运算能力足够强大，足以用于渲染虚拟现实世界。同时，手机的性能得到大幅提升。总之，当前的技术已经解决了 20 世纪 90 年代的许多局限。也正因如此，一些大型科技公司开始涉足于其中。

13.1.6 VR 的市场前景

很多专家预测，预计到 2020 年，全球头戴 VR 设备年销量会达 4000 万台，硬件市场规模

至少 400 亿元，加上内容、企业级市场，将是千亿以上。从长远来看，VR 产业规模万亿可期。

基于知觉管理与虚拟场景两大系统，我们从知觉捕捉、知觉反馈、主机、系统、应用和内容 6 大维度对虚拟现实的产业链进行解构。

1. 知觉捕捉设备

在各类知觉中，目前视觉捕捉是绝对主流，听觉、触觉捕捉尚不成熟，嗅觉、味觉捕捉还处于实验室阶段。视觉捕捉可以分为眼部追踪、头部追踪、肢体动作（手势等）捕捉、全身动作捕捉 4 种形式，不同的捕捉设备能够提供不同的沉浸感体验，也细分了捕捉设备市场。

2. 知觉反馈

视觉是人类获取信息的主要知觉系统，但目前视觉反馈设备尚不理想，主流的视觉反馈设备有眼镜、头盔、一体机 3 类，其中眼镜比较简陋、沉浸感不足，头盔和一体机沉浸感较好，但价格较高、便捷性较差。相比之下，听觉反馈已经相当成熟。

3. 主机

目前的 VR 内容主要通过移动端、PC 端或者一体机输入，脑电波计算仍在实验室阶段。所以，目前 VR 主机主要借助 PC、智能手机，也有不少公司将主机嵌入 VR 一体机中。总体来看，当前计算能力、存储空间、传输速率基本可以满足基础 VR 设备所需。由于 PC 的计算能力和扩展性强于手机，因此基于 PC 的 VR 头盔能够提供更好的体验。

支撑虚实不分的 VR 体验，目前的主机尚有 3 大局限。一是计算速度不足，虚拟出一个能够足以欺骗大脑的影像，而且可以和意识反馈互动，驱动这个影像的计算芯片超出现有普通的 PC 和手机。当然，提升终端计算力并不是目前科技发展的主流趋势，可以借助于速度越来越快的网络，将主要计算放在云端进行，而直接向终端下发计算结果。二是存储空间、传输速度、电池技术跟不上，VR 影像程序的体积以 10GB 为单位，如果直接从云端点播，我们需要 2~20MB/s 的下行速度——在目前的带宽环境下基本可以实现；但鲜有移动设备的电池能够支撑 20GB 大小数据量的持续高速下载，这决定了极致体验还需要依赖 PC 或专用主机，极大地限制了 VR 的使用场景。三是便捷性很差，PC 主机的体积和重量严重限制了它的使用场景，智能手机作为 VR 主机，也存在尺寸不匹配、散热等问题。

4. VR 系统

VR 系统即 VR 操作系统，是直接运行在主机上的系统软件，用于管理计算机硬件资源和软件程序、支持所有 VR 应用程序，是未来 VR 生态的基石。VR 系统的核心价值在于能够定义行业标准，搭建 VR 的基础和通用模块，无缝融合多源数据和多源模型。从产业格局上看，得系统者得平台、得行业话语权。

与其他领域类似，有先发优势的企业总愿意用封闭换体验，后来者则喜欢讲开源图颠覆，比如苹果 iOS 和谷歌 Android。在虚拟现实领域，Oculus 采用封闭的苹果模式已成定局，而 Google、雷蛇等后来者只能以开源、开放吸引开发者。

从国内来看，中小企业尚不具备开发 OS 的技术能力，大多希望从应用市场、播放器等出发打造平台；而互联网巨头们普遍在观望，等待最佳入局时机。

5. VR 应用

VR 应用分为 3 个层面：自上而下分别是应用软件、应用分发、中间件。

- 应用软件：提供各种场景下 VR 服务的软件，例如 VR 播放器、各类 VR 游戏等。应用软件是直接接触用户、决定用户体验的末端产品，是 VR 产业链软硬件技术的集中体现。目前，VR 应用有一些简单产品，随着硬件逐步成熟，将迎来大规模爆发。
- 应用分发：应用分发被认为是 VR 系统之外另一大入口。目前主流的应用分发平台有应用商店（移动端）和网站分发（PC）两大类，也有些 VR 论坛带有分发功能。由于目前 VR 行业目前还是硬件导向，VR 应用分发主要由硬件厂商主导。
- 中间件：是一种独立的系统软件或服务程序，可在不同系统间共享资源、可在不同应用中得到复用，典型的就是游戏引擎。成熟的 VR 中间件将促进标准统一，提升 VR 应用开发效率，快速引爆 VR 应用规模。已有一批中间件开始支持 VR 技术，英伟达（NVIDIA）2014 年 9 月发布了 VR Direct 技术，2015 年 8 月 26 日发布了 VR 游戏开发者的新开发套件——GameWorks VR Beta 版本，此外还有 SGI 的 OPENGL 接口、MS 的 DirectX 接口、AMD 的 Liquid VR 技术、Crytek 的 CryEngine、MultiGen-Paradigm 公司的 Vega Prime 等。

6. VR 内容

当前 VR 内容极为短缺，影视内容以短片和 UGC 为主，游戏几乎全是 DEMO。VR 内容将由一个个 CP（Content Provider）基于通用标准开发完成，VR 早期市场覆盖最大的产品一定是游戏内容和影视内容。国内外，几乎所有领域都在关注 VR，从影视、游戏到会展、直播、成人、旅游、地产等，其中不少企业已经投入研发制作。VR 内容总体分为三大方面：专业设备供应商、内容制作厂商、内容运营厂商。VR 典型设备，如 NEXTVR 的 VR3D 摄像机系统、诺基亚的 OZO、诺亦腾的全身动作捕捉系统等；内容制作环节，国外已经有不少著名影视业游戏公司参与进来，例如迪士尼、索尼等；国内大企业普遍还处于观望状态，反而是创业公司更为活跃，如 TVR 时光机、超凡视幻、兰亭数字、天舍文化、K-Labs、昊威创视等。

13.1.7 主流的硬件设备形态

VR 头戴设备（“眼罩”）主要分为 3 种：眼镜、头盔、一体机。

- 因为 PC 的局限以及 pc + 头盔使用场景的限制，VR 头盔也不太可能成为 2C 市场大规模普及的设备；但因为企业级客户对计算能力要求高、使用便捷性要求低，头盔会成为 B 端市场的主流设备。
- 由于智能手机性能持续快速提升，移动开发环境非常成熟和活跃，加上 VR 眼镜低成本带来价格优势，我们判断眼镜将是未来几年 VR 头戴设备的主流形态。
- 对于一体机来说，“轻便”与“性能”难以兼顾，而且价格较高。这也导致一体机不会成为近期的主流产品，世界主流 VR 厂商目前都还没有推出一体机。但我们相信，随着技术进步和元件微型化，VR 一体机将在性能、轻便上实现兼顾，而且以低于头盔、高于眼镜的价格赢得广泛用户。

13.1.8 谁会领衔 VR 内容制作

由于处于投入期，整个 VR 内容行业都附着在 VR 硬件产业上。

在早期，VR 内容不具备盈利条件，所以内容公司动力不足；相反，VR 硬件公司为了教育市场，必须有内容支撑才能提供完整体验，所以目前是硬件行业推动内容建设。例如，领军者 Oculus 收购游戏代码引擎 RakNet、3D 场景技术公司 Surreal Vision、成立电影工作室 Story Studio，陆续推出标杆式 VR 电影短片《Lost》《Henry》《Help》等，做了整个产业链该做的事。国内的暴风科技、3Glass、乐蜗等也包揽了应用分发、视频、游戏等环节，其中应用分发普遍自建平台，而视频游戏内容多是网络下载和对外合作，少部分自主研发。

随着 VR 头戴设备的普及，VR 内容分发将独立发展，最终成为行业入口。随着行业逐渐发展、内容日趋丰富、版权趋于规范，用户在一家硬件公司获得的内容将非常有限，而硬件公司做应用分发则更加不经济和不效率，所以 VR 应用分发会逐渐成为一个独立产业环节，而覆盖更多头戴设备和用户的平台将掌握这一行业入口。我们分析搭配手机使用的 VR 眼镜会成为近期的主流设备，所以掌握 VR 应用分发话语权的有可能是主流手机厂商或者广泛兼容各型手机的应用分发平台。

13.2 基于 Unity3D 的 Android 平台 VR 应用开发

北京时间 2015 年 5 月 29 日凌晨 0:30 在美国旧金山举办的 2015 谷歌 I/O 开发者大会上，素以慷慨著称的谷歌并没有像以往那样大派礼物，除了三星或者 LG 智能手表的二选一外，开发者还可以领到一个小小的黄色纸板盒 Cardboard，如图 13-1 与图 13-2 所示。不过，这个看起来非常寒碜的再生纸板盒却是 I/O 大会上最令人惊喜的产品，这就是谷歌推出的廉价 3D 眼镜。本书中的 VR 应用就是将 Android 手机和 Google Cardboard 结合，并使用 Cardboard SDK 在 Android 手机上开发出 Cardboard 应用来实现的。





图 13-1　Cardboard 后视图



图 13-2 Cardboard 测试图

Cardboard 最初是谷歌法国巴黎部门的两位工程师大卫 • 科兹（David Coz）和达米安 • 亨利（Damien Henry）的创意。他们利用谷歌“20%时间”规定，花了 6 个月的时间，打造出来这个实验项目，意在将智能手机变成一个虚拟现实的原型设备。

Cardboard 纸盒内包括了纸板、双凸透镜、磁石、魔力贴、橡皮筋以及 NFC 贴等部件。按照纸盒上面的说明，几分钟内就组装出一个看起来非常简陋的玩具眼镜。凸透镜的前部留了一个放手机的空间，而半圆形的凹槽正好可以把脸和鼻子埋进去。

Cardboard 只是一副简单的 3D 眼镜，但这个眼镜加上智能手机就可以组成一个虚拟现实（VR）设备。

要使用 Cardboard，用户还需要在 Google Play 官网上搜索 Cardboard 应用。它可以将手机里的内容进行分屏显示，由于两只眼睛看到的内容有视差，因此会产生立体效果。通过使用手机摄像头和内置的螺旋仪，在移动头部时能让眼前显示的内容产生相应变化。应用程序可以让用户在虚拟现实的情景下观看 YouTube、谷歌街景或谷歌地图。

CardBoard 的虚拟现实效果是由一款 CardBoard 与一部安卓手机结合而成的，眼镜镜体通过透镜加屏幕的原理，将虚像呈现在人的明视距离处，实现了沉浸式的虚拟现实感，目前国内虚拟现实眼镜（如暴风魔镜等）大都是这个原理，只不过做了细致的包装，使得佩戴更加舒适。图 13-3 所示即为其原理图（可根据原理自行 DIV）。

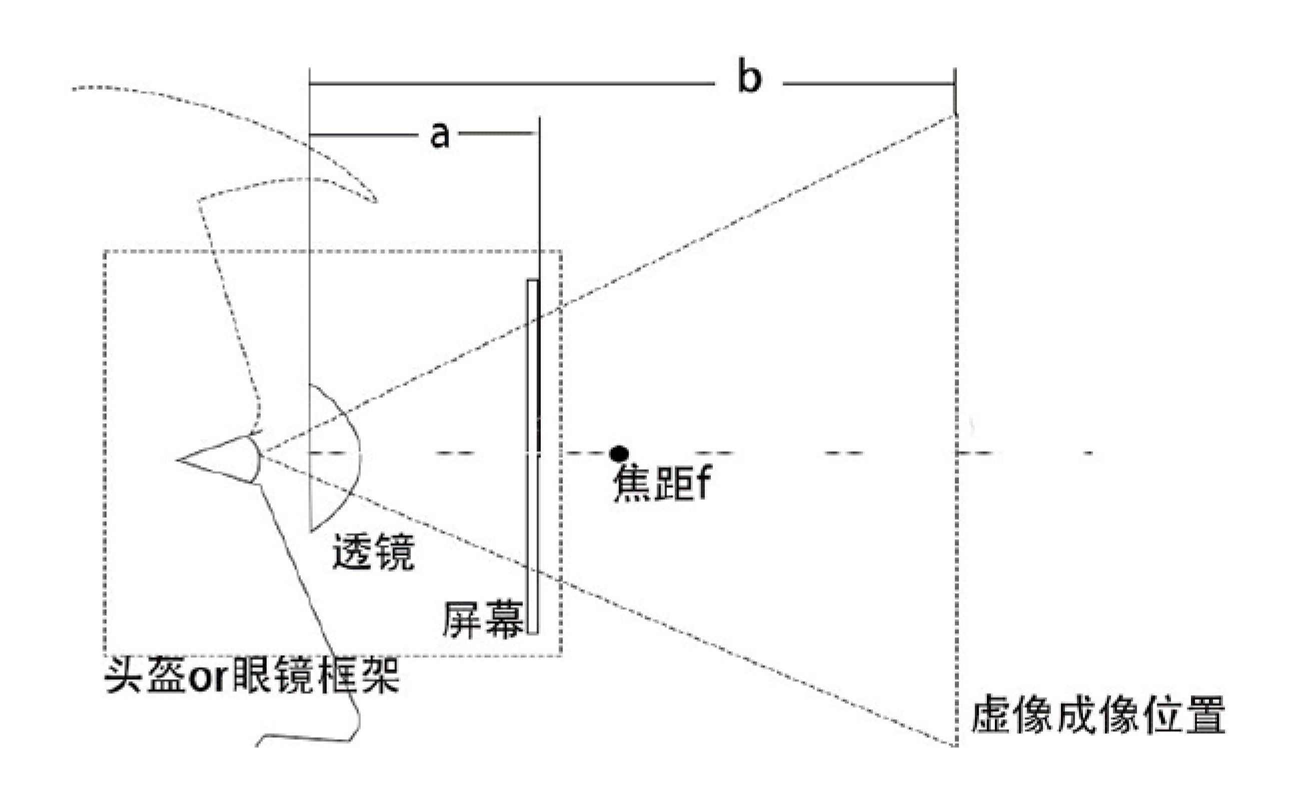


图 13-3　Cardboard 实现原理图

在 Android 平台上的 CardBoard 应用开发有两种方式：

- 第一种方式是通过 Google 提供的 Cardboard SDK，工程师可以使用 Android 原生的 SurfaceView 与 OpenGL ES 开发。这种方式可扩展性很强，但是相应的复杂度比较高，导入 3D 模型等都需要手写代码。
- 另外一种在 Android 平台上进行 VR 开发的方法是通过 3D 引擎（如 Unity3D 等）进行开发。这种方式适合开发游戏，复杂性较低，模拟左右双眼只需要两个摄像机就可以了。Unity 引擎功能强大，基本上能适应大部分需求，而且开发便利，资料很全。Google 提供了一个 Cardboard SDK for Unity，可以很方便地进入虚拟现实的世界，但是开发复杂应用又会力不从心。

本书将讲述如何通过 Unity3D 进行 VR 开发，如果想要使用 Android 原生的 SurfaceView 与 OpenGL ES 开发，也可以下载 Google 官方提供的 demo 进行研究。本书是描述 Android 开发的书籍，对 VR 的讲述主要是希望通过本章内容使读者可以入门，同时涉及的相关 Unity3D 知识非常简单，所以关于 Unity3D 开发并不进行深入讲解。如有兴趣，读者可自行研究。文中

将通过利用 Google 官方的 demo 创建一个自己的场景，把自己的模型放进场景，用虚拟现实眼镜进行观赏甚至操作。

13.2.1 下载 Cardboard SDK for Unity

进入 Google Cardboard 官方网站的开发者指南页面，如图 13-4 所示。点击左侧 Unity SDK 下的 Download and Samples（https://developers.google.com/cardboard/unity/download），再点击 Download Cardboard SDK for Unity (direct link to zip)进行下载。

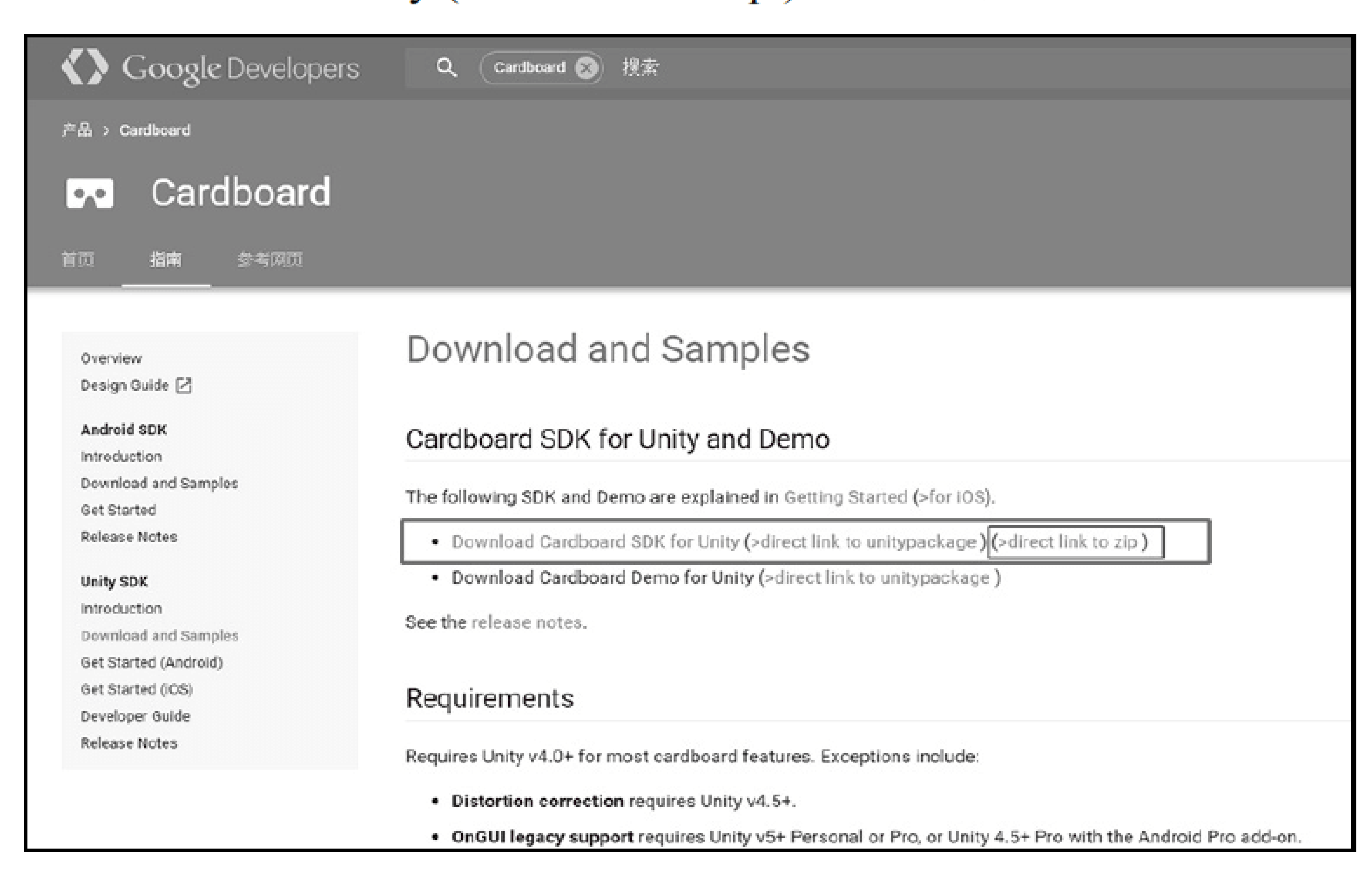


图 13-4 Cardboard 官方网站的开发者指南页面

13.2.2 导入 CardboardSDKForUnity.unitypackage

如果下载的是旧版 SDK 包，那么里面只有一个 CardboardSDKForUnity.unitypackage，导入之后包含支持代码和一个例子；新版 SDK 包中则包含 CardboardSDKForUnity.unitypackage 和 CardboardSDKForUnity.unitypackage 两个包，第一个是库，第二个是 Demo，将两个包都导入后即可运行实例。

首先打开 Unity，新建一个 Project，如图 13-5 所示。

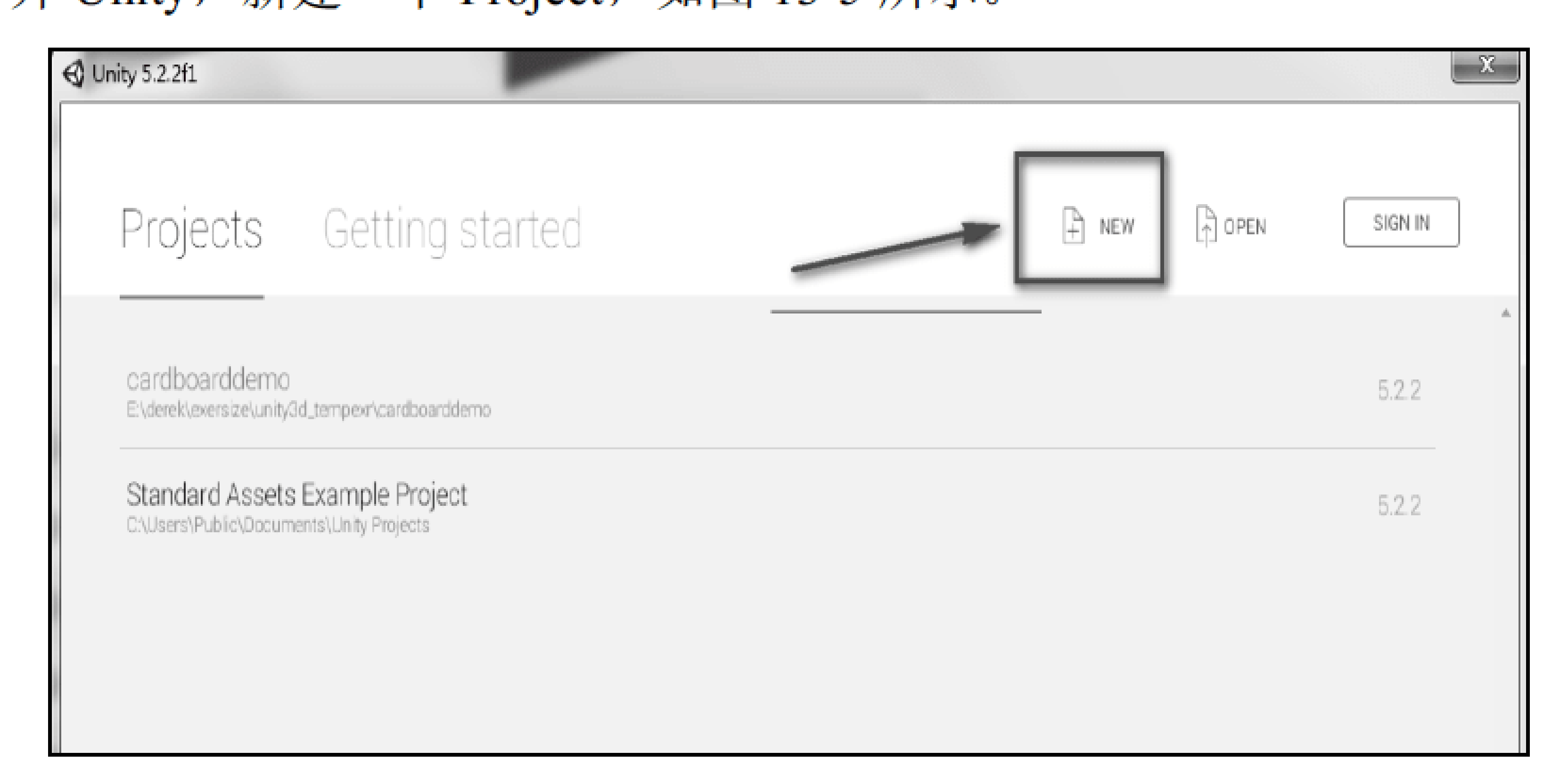


图 13-5 打开 Unity 并新建一个工程

然后选择 Assets→Import Package→Custom Package…，如图 13-6 所示。

图 13-6　打开 Custom Package

引入下载好的 SDK 包，如图 13-7 所示。

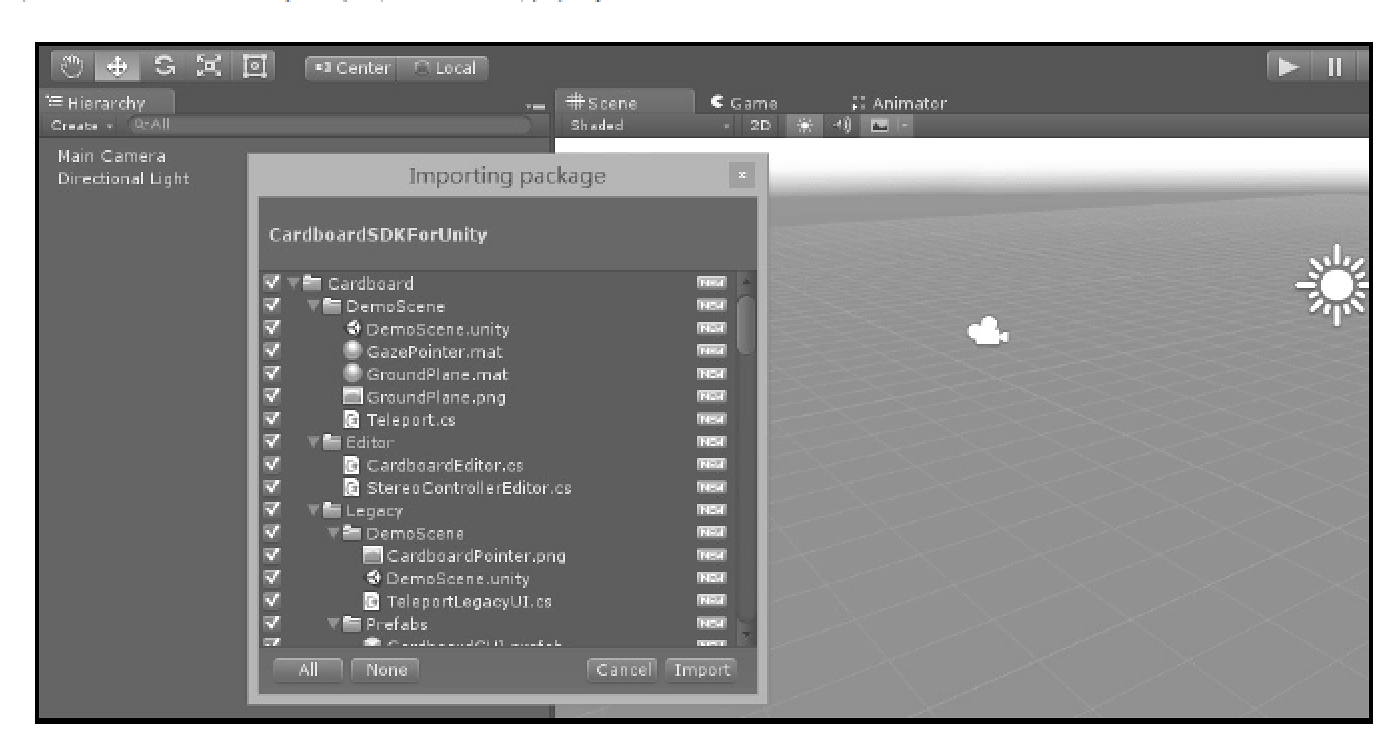


图 13-7　引入 SDK 包

13.2.3　运行 DemoScene

把.unitypackage 文件导入之后，在 Project 面板的资源文件夹下就会多出一个 Cardboard 文件夹，包括 SDK 的插件代码和 Demo 示例。查看 Cardboard 文件夹下的 DemoScene 文件夹，这是其中的一个示例，如图 13-8 所示。

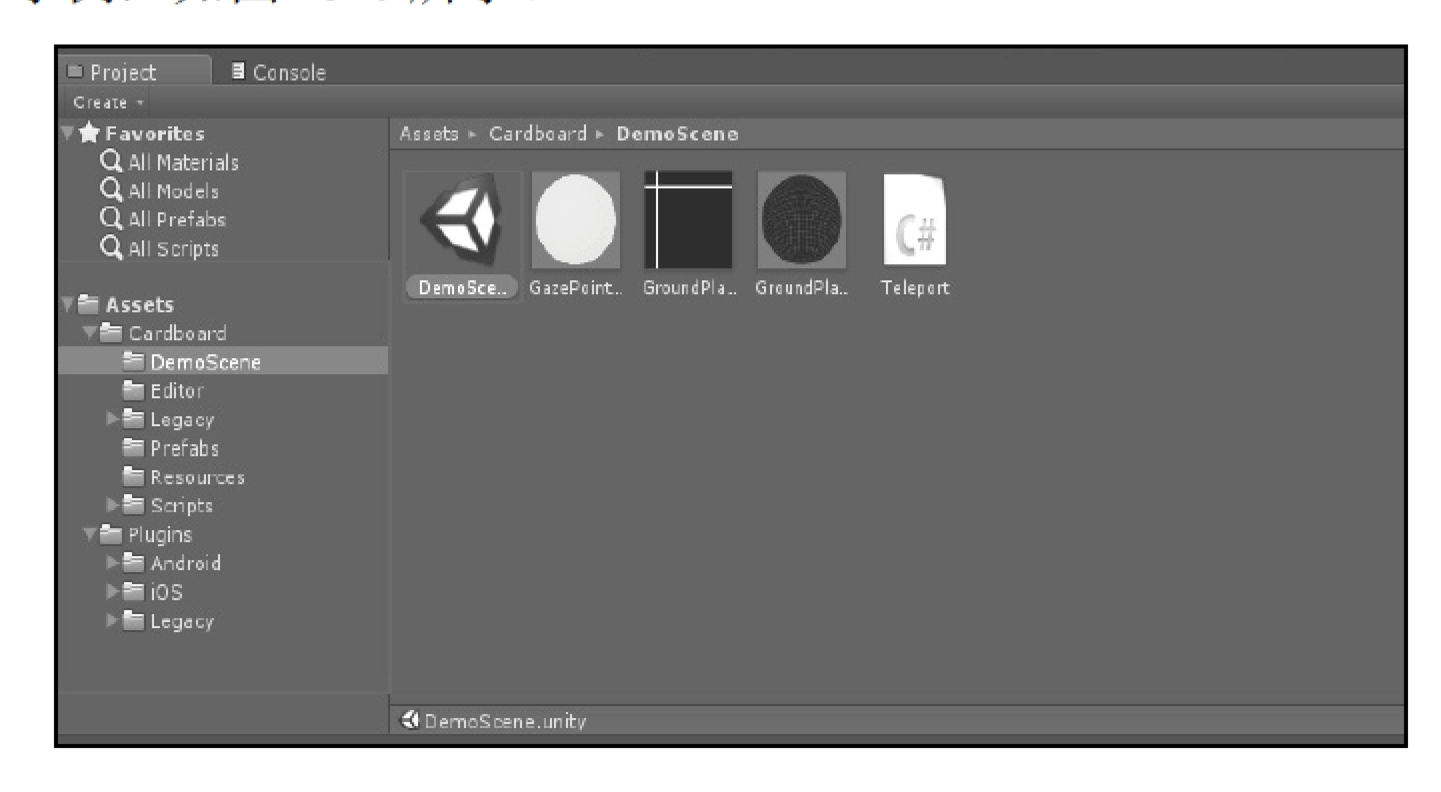


图 13-8　DemoScene 实例文件夹中的内容

双击场景文件 DemoScene，打开示例，如图 13-9 所示。

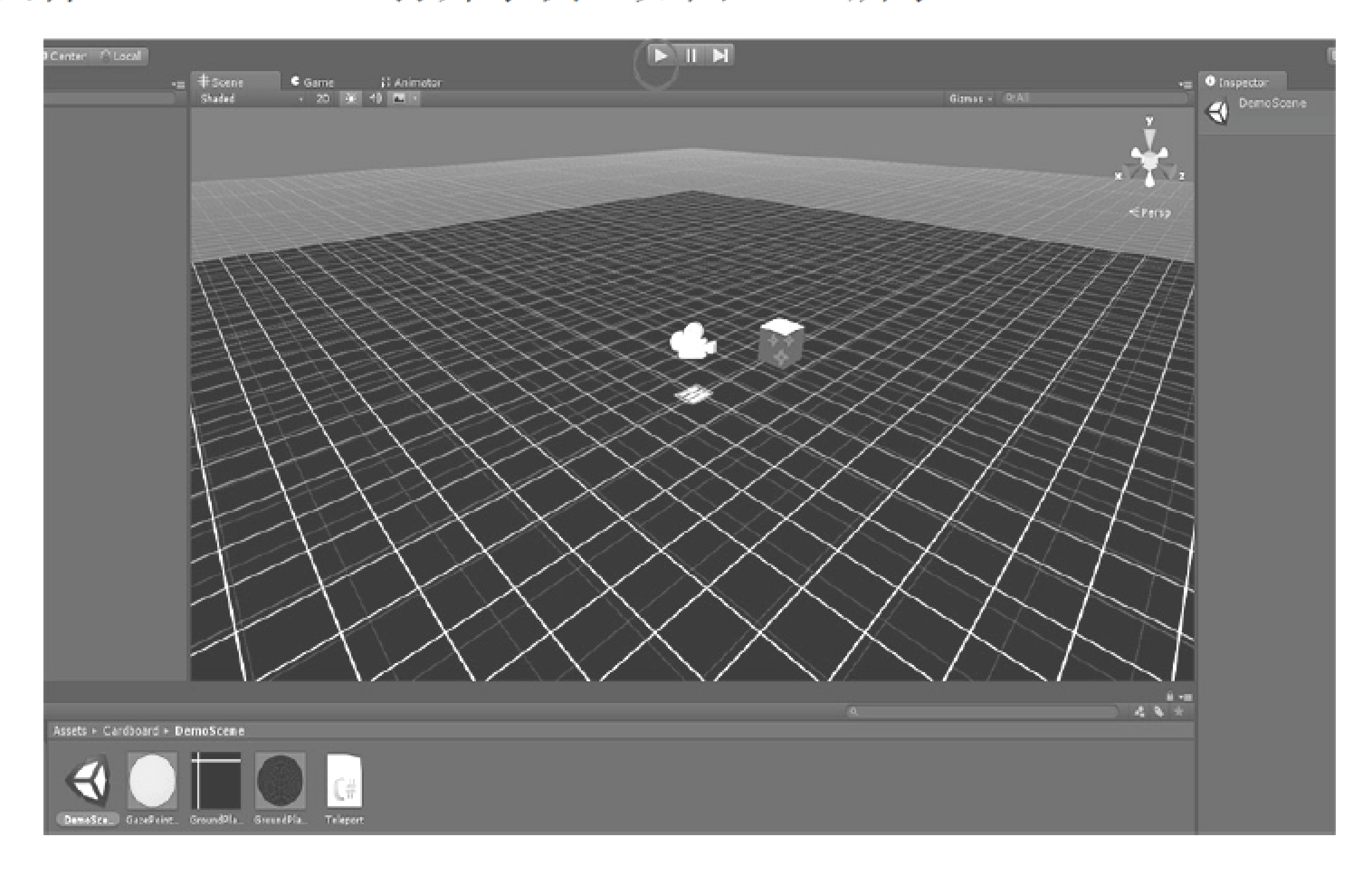


图 13-9　在 Unity 中打开场景文件 DemoScene 效果图

点击上方的运行按钮（小箭头），就可以看到 Demo 示例的运行效果了，如图 13-10 所示。

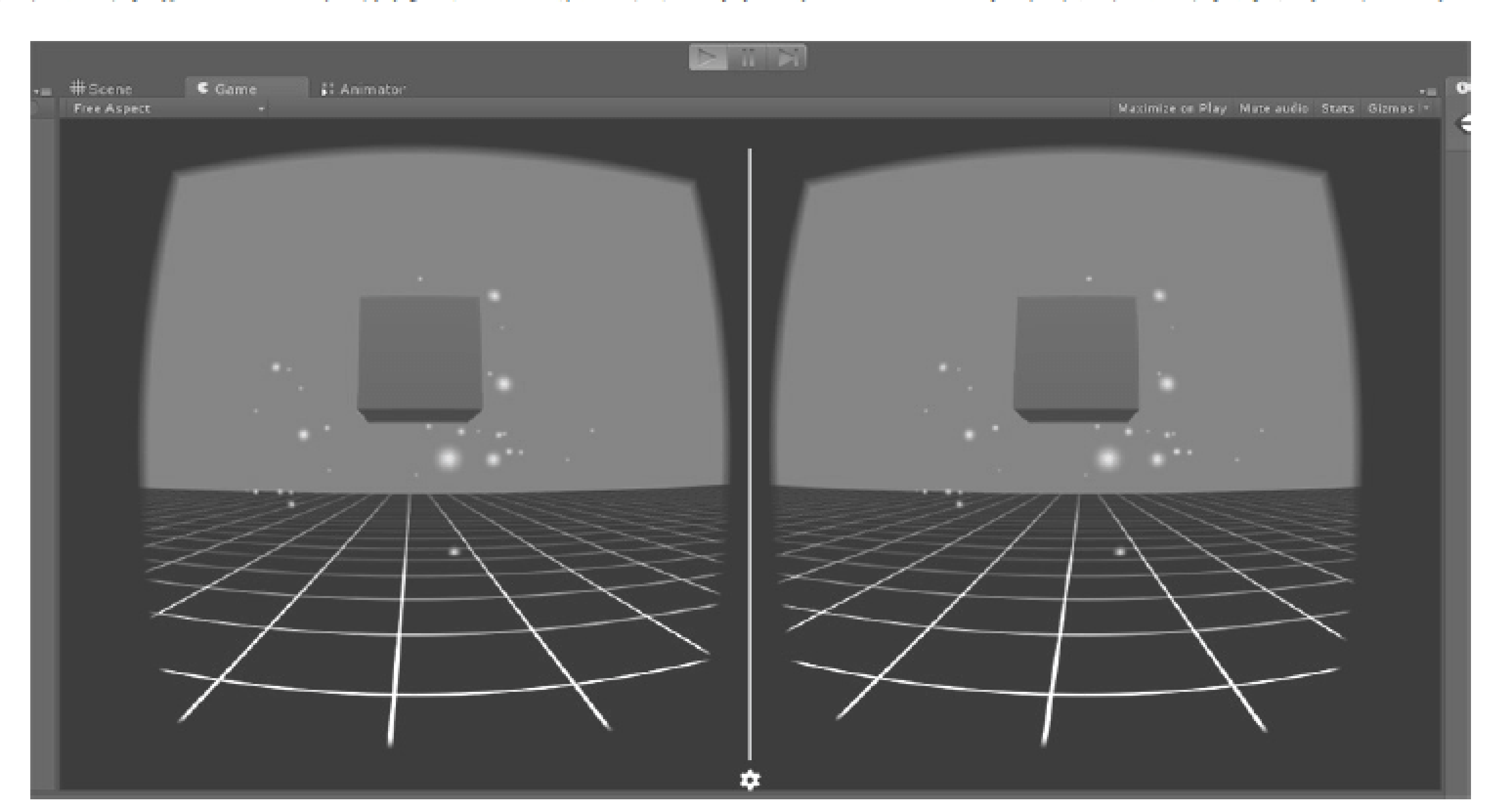


图 13-10　DemoScene 实例的运行效果

运行之后，按住 Alt 键移动鼠标，模拟头部转动，按住 Ctrl 键模拟歪脖子的时候视角会变化，点击鼠标相当于触发，可以用来操作。这个 Demo 且有几个功能：①把目光也就是小黄点对准方块，点击鼠标，方块会转动到一个有距离限制的球面上的随机位置。②当目光注视方块时，方块会从红色变成绿色，当目光离开方块时，方块会从绿色变回红色。③在脚下有 3 个按钮，分别是下面 3 种：Reset，重新把方块放回初始位置； Recenter，重新把视角左右方向上回归中间；VR Mode，打开或者关闭 VR 模式（分屏与否）。

这个 Demo 的代码只有一个文件，并且十分短小，仅仅几行脚本就实现了分屏、陀螺仪、视角转动等功能。我们不得不说，Cardboard SDK 功能还是十分强大的。接下来设置 Android SDK 路径，打包导出为安卓工程，在手机上安装。

需要强调的是，必须安装了 Android 支持插件之后才能设置 Android SDK，否则是设置不了的，如图 13-11 所示。互联网上的很多教程都忽略了此步骤，让很多初学者吃了苦头。

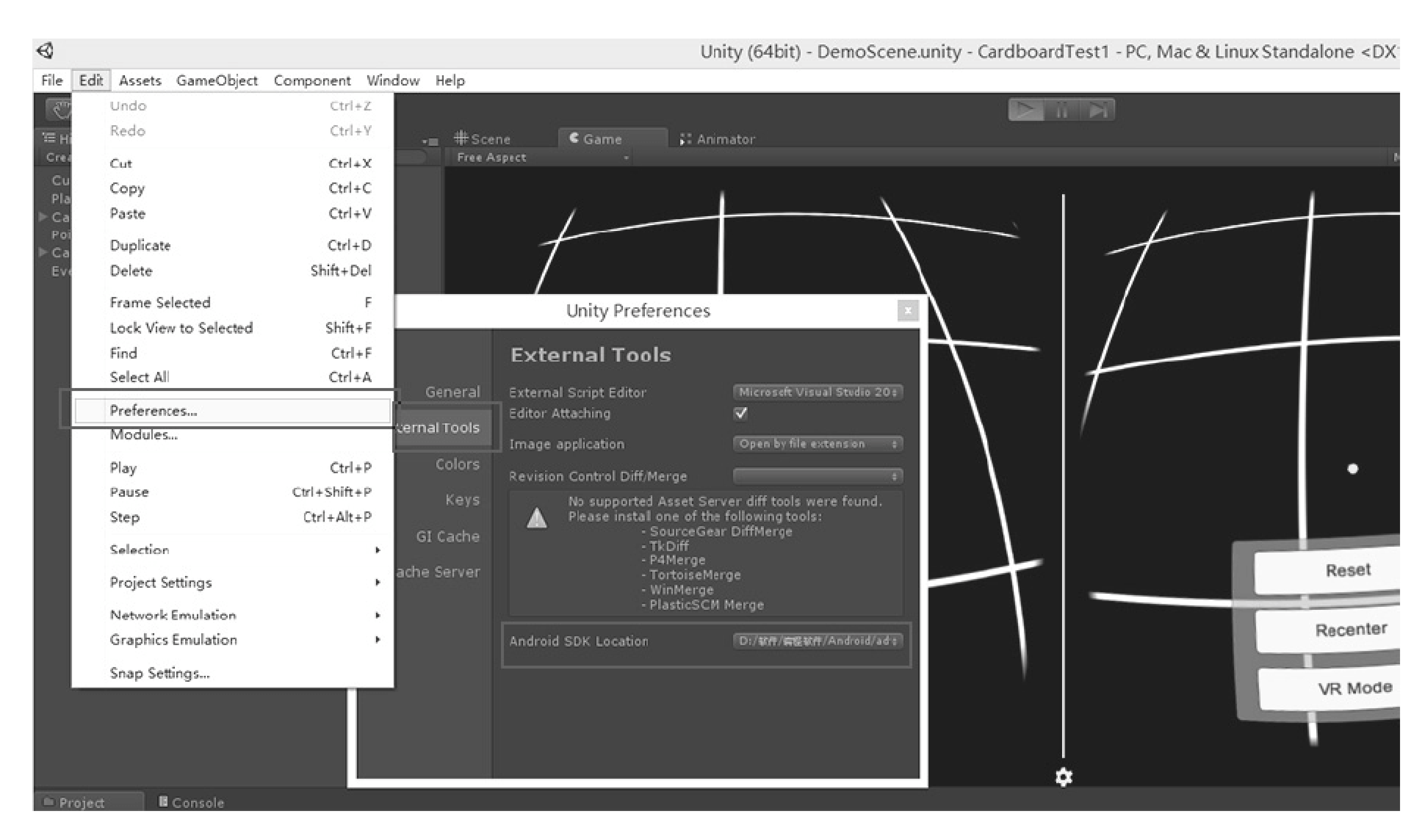


图 13-11　设置 Android SDK

导出 Android Apk，如图 13-12 所示。这里需要说明的是，在导出时需要进行一些设置，点击下面的 Player Settings，之后右侧会出现设置界面，如图 13-13 所示。这里需要重新设置一下包名，使用默认包名会打包失败，还可以设置应用的图标、名称等。

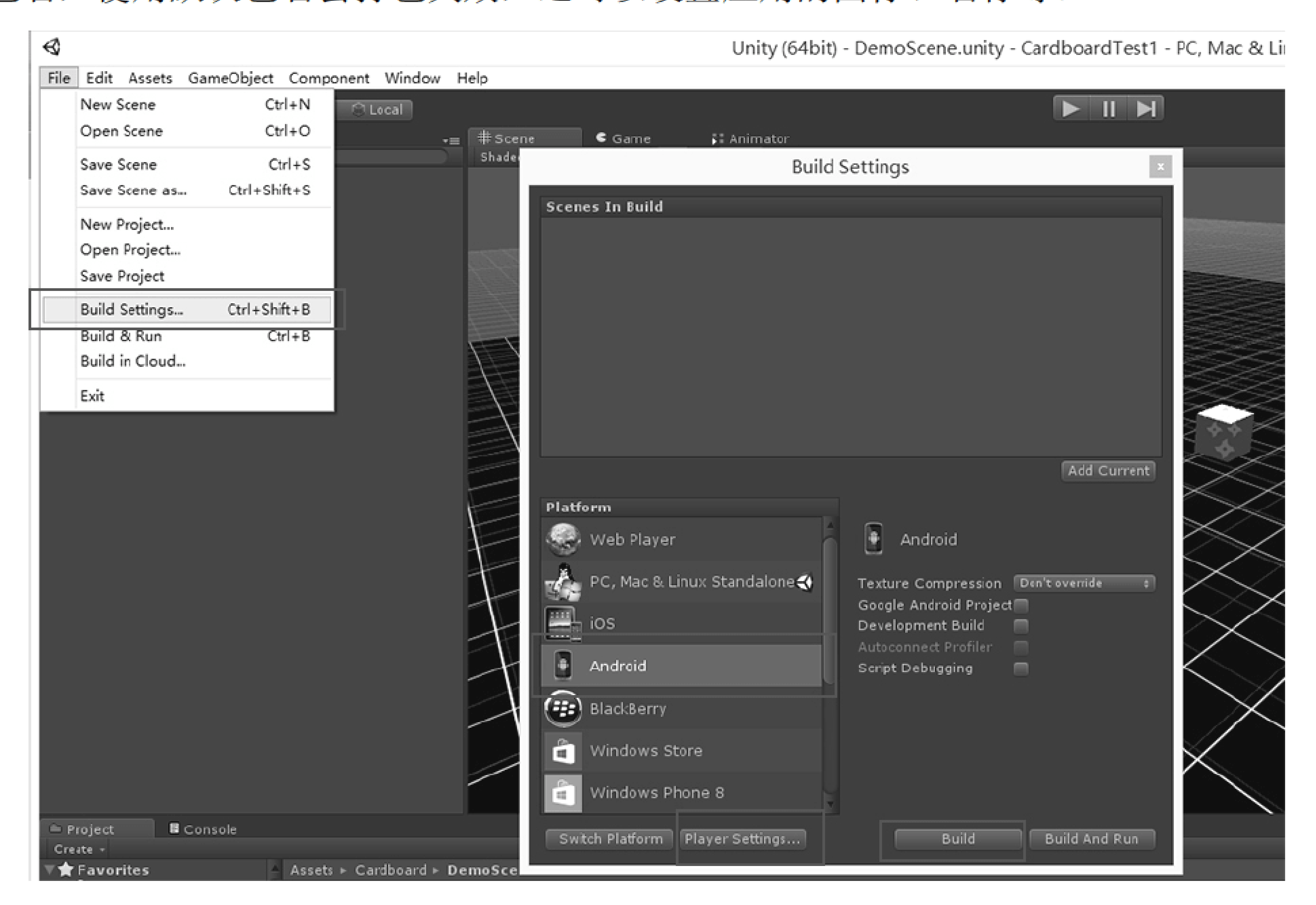


图 13-12　导出 Android Apk 文件

安装到手机（也可以放在 Cardboard 或者暴风魔镜等成品镜中进行感受。如果条件不允许，直接拿起手机横屏观看也可以。）之后的效果如图 13-14 所示。

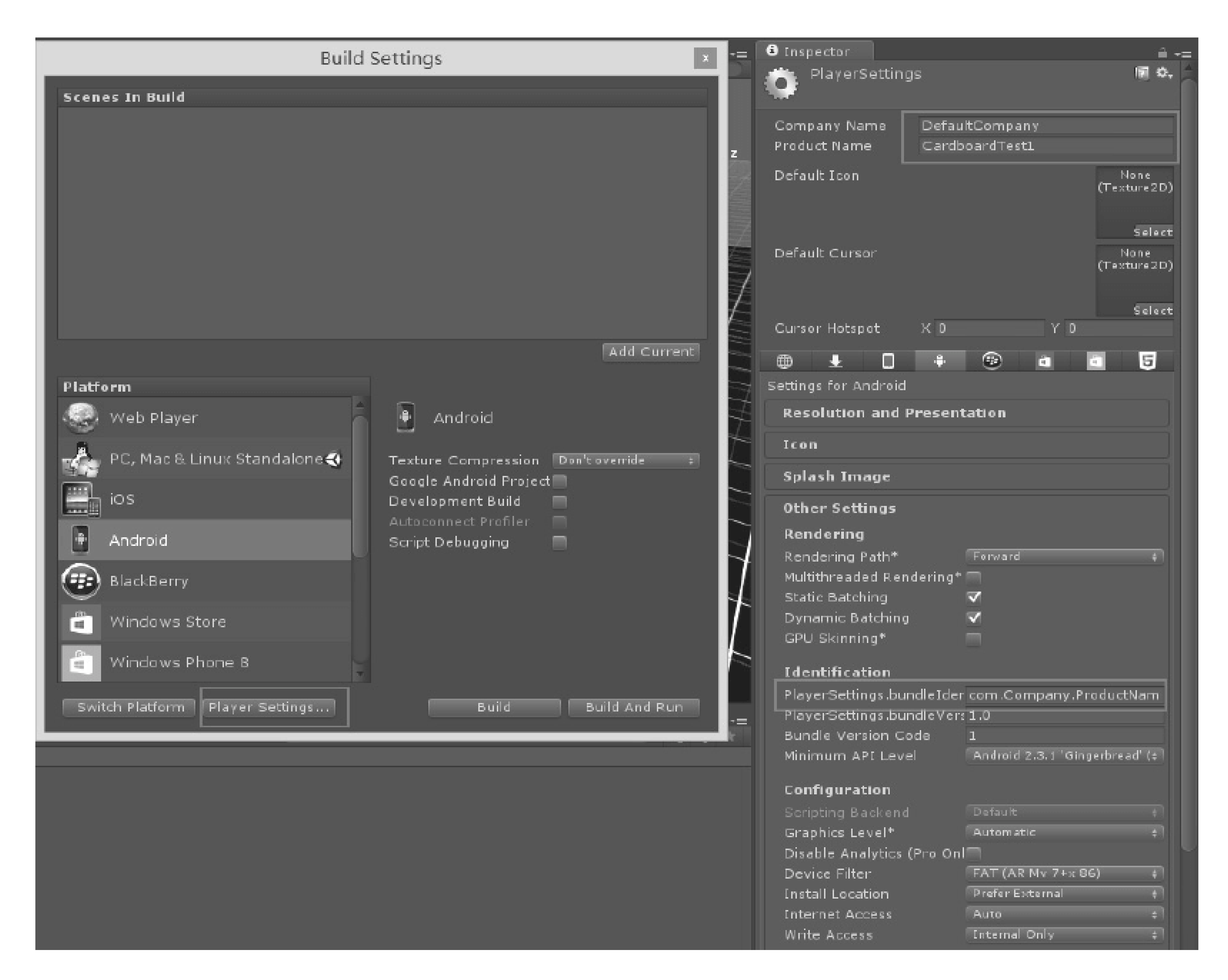


图 13-13　导出 Android Apk 文件之前做的一些设置

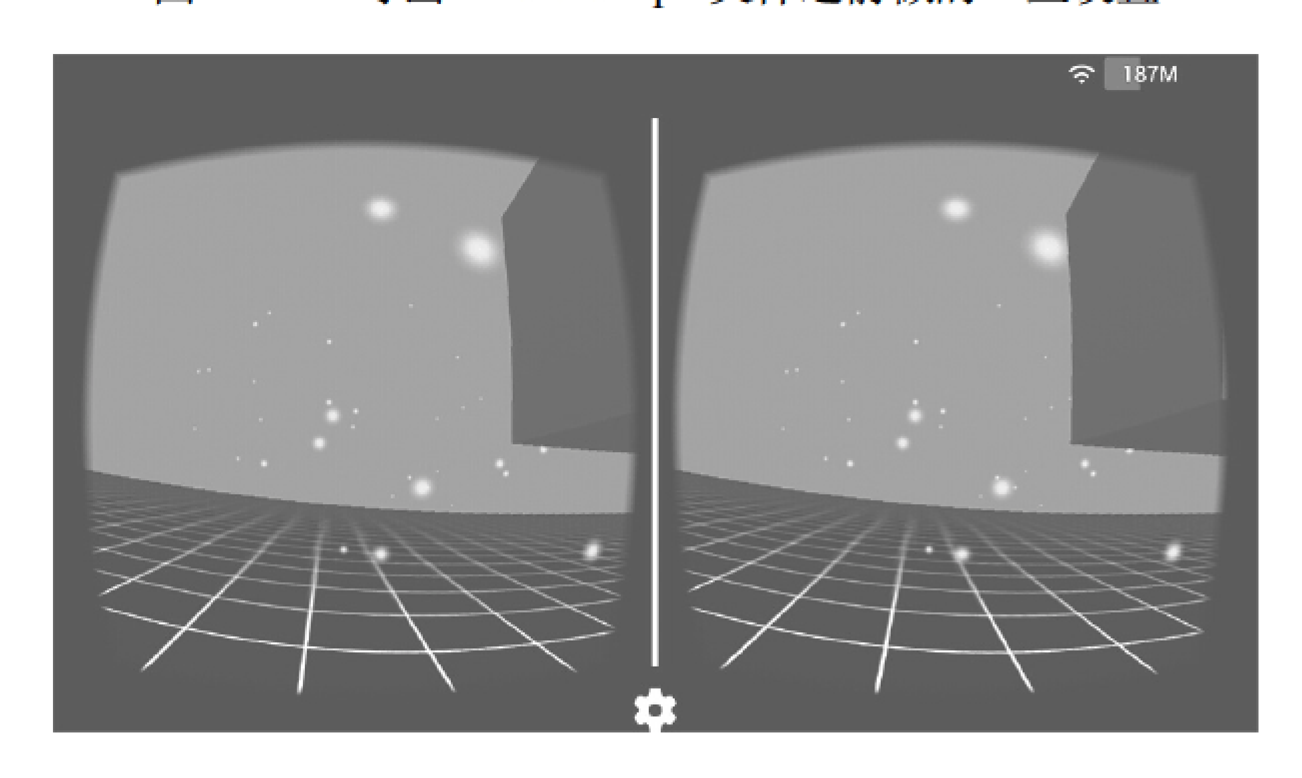


图 13-14　在手机中观察制作的 VR 内容的效果

13.2.4　使用 Unity3D 创建一个自己的场景

使用 Unity3D 创建一个自己的场景其实很简单，只需要从网上下载一份 FBX 格式的渲染图，直接拖进 Unity3D 即可。图 13-15 所示是下载的 FBX 格式 3D 图。



图 13-15　下载的 FBX 格式 3D 图

点击左侧的 CardboardMain，也就是左右眼摄像机组成的主摄像机，用移动工具把它移动到想要的位置，再把摄像机放置到了机舱内部，模拟驾驶员视角。制作过程如图 13-16 所示，运行之后的效果如图 13-17 所示。

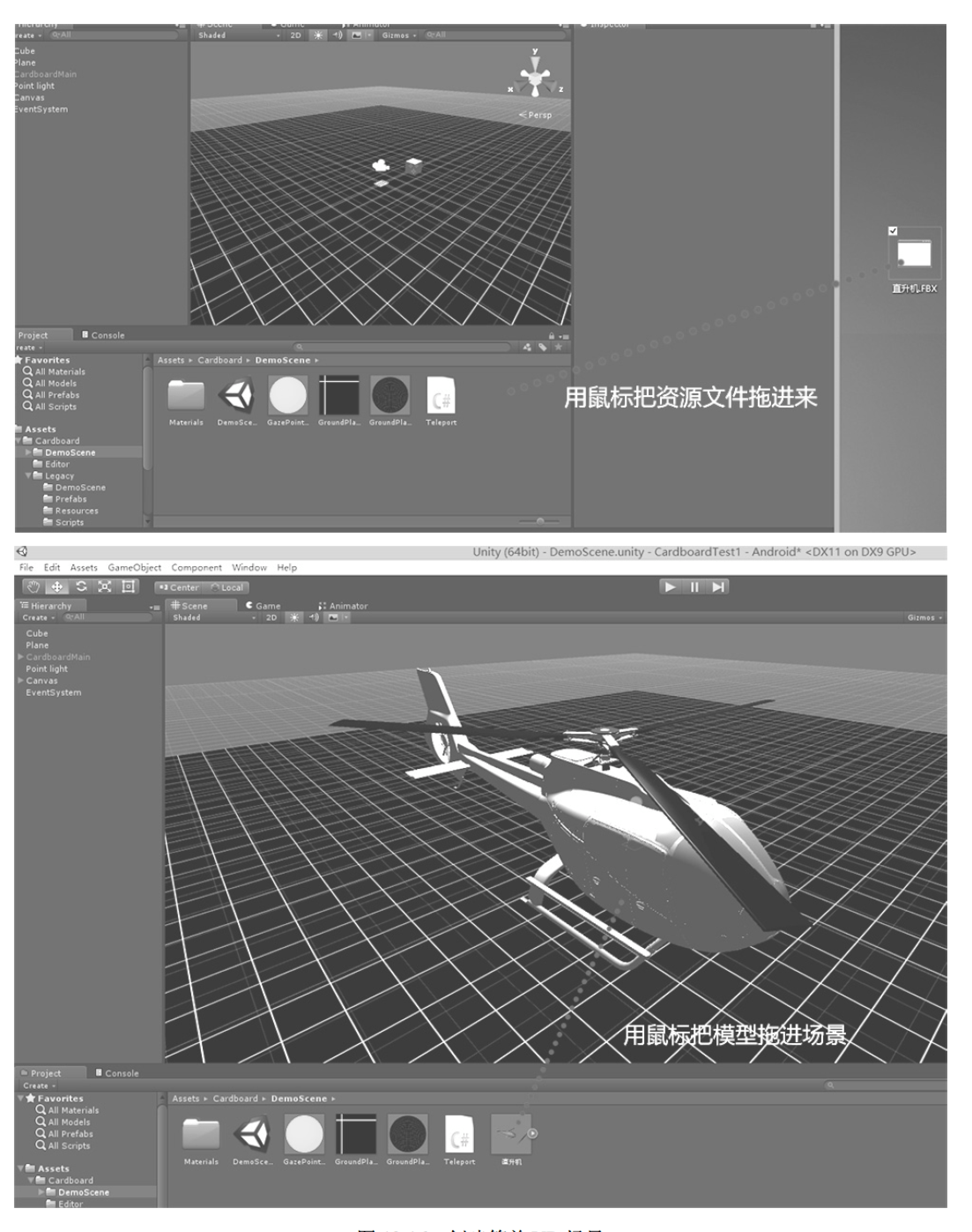


图 13-16 创建简单 VR 场景

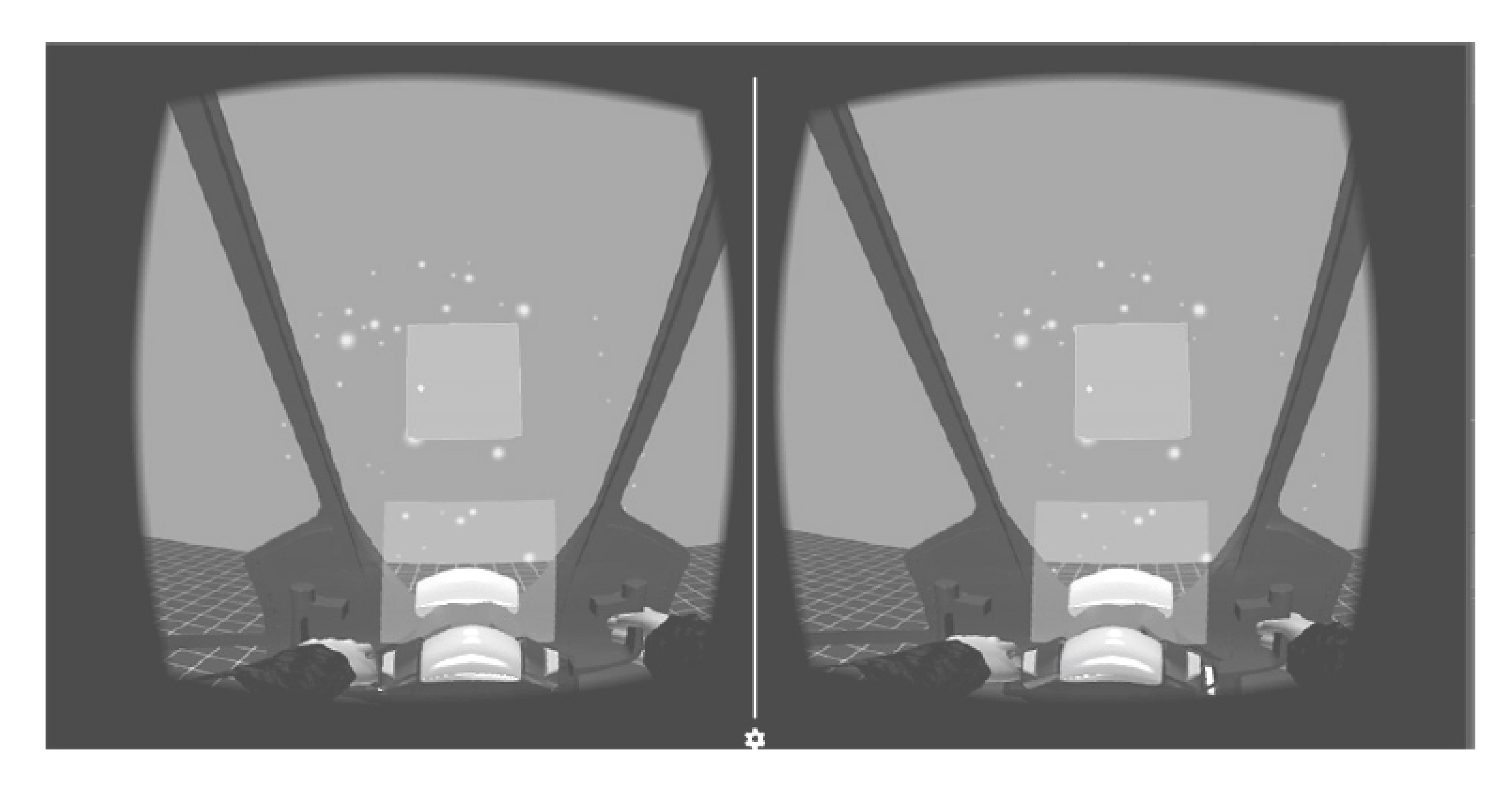


图 13-17　自己创建的简单 VR 场景的运行效果

这里打包成 Apk 文件的过程和上述过程一致，不再赘述。

13.3　小　　结

VR 作为全球科技圈最热门的新技术、新领域，同时也是一个全新的消费领域，当之无愧成为创新的主角，自然也受到了各方资本的热捧。目前，国内出现了不少 VR 创业公司，产业还处于启动期，涉及 VR 设备（眼镜等）、内容制作（游戏、视频等）、发布平台等，大量的头戴眼镜盒子、外接式头戴显示器等 VR 设备向消费级市场拓展，在政策的扶持、资本的推动下，自 2015 年以来，参与虚拟现实领域的企业大幅增加，目前国内有超过 100 家 VR 设备开发公司。同时，在我国创新驱动战略的推动下，包括许多 VR 创业公司在内，我国的 VR 行业吸引了大量的资本注入。我国 VR 行业呈现出了欣欣向荣的发展景象。此时进入 VR 行业正当其时。关于 VR 应用的开发，本章只是提供了简单的案例，之后还会有相关书籍出版，以供读者学习。读者也可以下载 carbon for Android 的 SDK 进行学习。



第 14 章 Android NDK 开发入门

Java 是半解释型语言，很容易被反汇编后拿到源代码文件，在开发一些重要协议时，为了安全起见，使用 C 语言来编写重要的部分，增大系统的安全性。那么 C 语言编写的程序该如何编译进 Android 应用中呢？这就是本章将要讲解的重点——NDK 开发。本章讲解的 NDK 开发是基于 Android Studio 进行的，和基于 Eclipse 进行的有很大不同。以往的很多书籍中都是以 Eclipse 作为 IDE 进行开发的，而现在 Google 不再支持 Eclipse，所以讲解如何使用 Android Studio 进行 NDK 开发就变得尤为重要。

14.1　NDK 简介

Android 平台从诞生起就支持 C、C++开发。众所周知，Android 的 SDK 基于 Java 实现，这意味着基于 Android SDK 进行开发的第三方应用都必须使用 Java 语言，但这并不等同于“第三方应用只能使用 Java”。在 Android SDK 首次发布时，Google 就宣称其虚拟机 Dalvik 支持 JNI 编程方式，也就是第三方应用完全可以通过 JNI 调用自己的 C 动态库，即在 Android 平台上“Java+C”的编程方式是一直都可以实现的。

不过，Google 也表示，使用原生 SDK 编程相比 Dalvik 虚拟机也有一些劣势，Android SDK 文档里找不到任何 JNI 方面的帮助。即使第三方应用开发者使用 JNI 完成了自己的 C 动态链接库（so）开发，so 又如何和应用程序一起打包成 Apk 并发布呢？这里面也存在技术障碍，比如程序更加复杂、兼容性难以保障、无法访问 Framework API、Debug 难度更大等。开发者需要自行斟酌使用。

于是 NDK 就应运而生了。NDK 的全称是 Native Development Kit。

NDK 的发布使“Java + C”的开发方式转正，成为官方支持的开发方式。NDK 将是 Android 平台支持 C 开发的开端。

NDK 提供了一系列的工具，能够帮助开发者快速开发 C（或 C++）的动态库，并能自动将 so 和 Java 应用一起打包成 Apk。这些工具对开发者的帮助是巨大的。NDK 集成了交叉编译器，并提供了相应的 mk 文件隔离 CPU、平台、ABI 等差异，开发人员只需要简单修改 mk 文件（指出“哪些文件需要编译”“编译特性要求”等）就可以创建出 so。NDK 可以自动将 so 和 Java 应用一起打包，极大地减轻了开发人员的打包工作。

NDK 提供了一份稳定、功能有限的 API 头文件声明。Google 明确声明该 API 是稳定的，在后续所有版本中都稳定支持当前发布的 API。从该版本的 NDK 中可以看出，这些 API 支持的功能非常有限，包含 C 标准库（libc）、标准数学库（libm）、压缩库（libz）、Log 库（liblog）。

使用 NDK 开发有如下优势：Apk 的 Java 层代码很容易被反编译，而 C/C++库反汇难度较大；可以方便地使用现存的开源库，大部分现存的开源库都是用 C/C++代码编写的；用 C/C++写的库可以方便在其他嵌入式平台上再次使用。

这里做一下说明：NDK 并不能显著提升应用效率。很多读者可能会觉得 C 语言比 Java 效率要高得多，但是随着 jdk 的不断更新，Java 的效率也逐渐提高；另外，即便使用 C 语言编码提高了应用效率，在 Java 与 C 相互调用时也会增大开销。

14.2　使用 Android Studio 进行 NDK 开发

本节内容将和读者开发一个 Android NDK 的实例，帮助读者掌握 Android NDK 开发。

14.2.1 Android NDK 开发环境搭建

进行 Android NDK 开发时，需要下载 Android NDK 开发工具包。只需要打开 Android Studio，进入 SDK Manger，选择 SDK Tools，在下面的选项中选中 Android NDK 选项，点击"Apply"，Android Studio 就会自动下载 Android NDK，如图 14-1 所示。

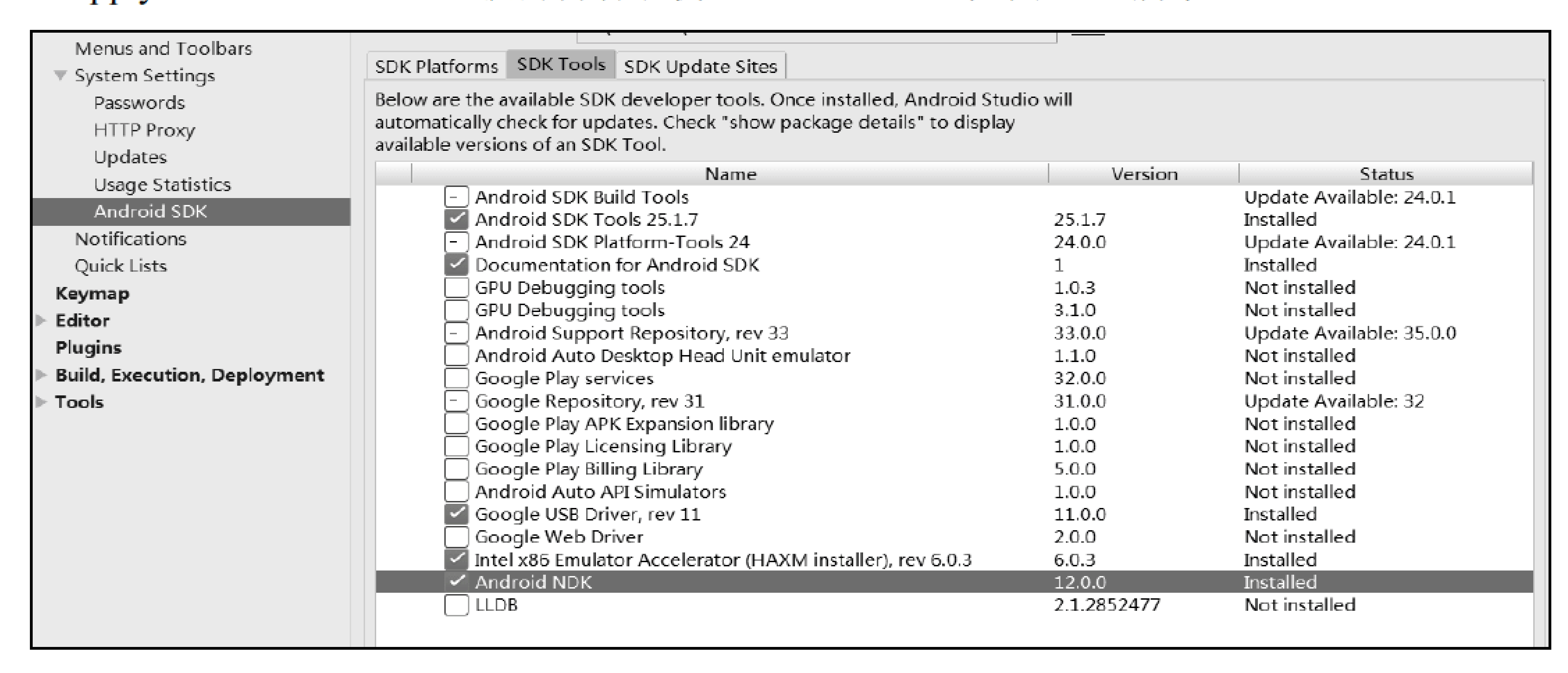


图 14-1 下载 NDK 开发工具包

这个下载过程可能会比较长，下载完成后，点击 OK 就可以了。这时 Android NDK 就下载配置完成了，可以新建一个 Android NDK 工程了。

新建一个 Android 工程，打开 local.properties 文件，会发现：

```
ndk.dir=D\:\\android\\sdk\\ndk-bundle
sdk.dir=D\:\\android\\sdk
```

这两行分别是系统自动根据我们下载时的路径配置好的 Android SDK 路径和 Android NDK 路径。

要想在 Android Studio 中进行 Android NDK 开发，还需要修改 3 个文件，分别是项目目录下的 build.gradle 文件、gradle 文件夹下 warpper 文件夹下的 gradle-wrapper.properties 文件、app 目录下的 build.gradle 文件。

项目目录下的 build.gradle 文件修改如下：

```
buildscript {
    repositories {
        jcenter()
    }
    dependencies {
        classpath 'com.android.tools.build:gradle-experimental:0.2.0'
    }
}
allprojects {
```

```
        repositories {
            jcenter()
        }
}
task clean(type: Delete) {
        delete rootProject.buildDir
}
```

gradle-wrapper.properties 文件修改如下：

```
distributionBase=GRADLE_USER_HOME
distributionPath=wrapper/dists
zipStoreBase=GRADLE_USER_HOME
zipStorePath=wrapper/dists
distributionUrl=https\://services.gradle.org/distributions/gradle-2.5    -all.zip
```

app 目录下的 build.gradle 文件修改如下：

```
apply plugin: 'com.android.model.application' //新加的 model
model {
        android {
            compileSdkVersion = 23 //这些是软件根据选择的版本自动写的，不必改
            buildToolsVersion = "23.0.1"//这些是软件根据选择的版本自动写的，不必改
            defaultConfig.with {
                applicationId = "lrq.buaa.com.ndk" //这是程序包名，用你自己的
                minSdkVersion.apiLevel = 15
                targetSdkVersion.apiLevel = 23
                versionCode = 1
                versionName = "1.0"
            }
            tasks.withType(JavaCompile) {
                sourceCompatibility = JavaVersion.VERSION_1_7
                targetCompatibility = JavaVersion.VERSION_1_7
            }
        }
        android.ndk {
            moduleName = "lb" //这是将来 so 文件的名称，可以自己取
        }
        android.buildTypes { //从原来的 android 模块中移出来的
            release {
                minifyEnabled = true
                proguardFiles.add(file("proguard-rules.pro"))              }
        }
}
dependencies {
```

```
        compile fileTree(dir: 'libs', include: ['*.jar'])
        compile 'com.android.support:appcompat-v7:23.4.0'
    }
```

这里就不展示修改前的内容了，读者可以自行建立一个 Android 工程比对分析。仔细分析上面 3 个需要修改的文件会发现项目目录下的 build.gradle 文件只修改了“classpath 'com.android.tools.build:gradle-experimental:0.2.0'”，gradle-wrapper.properties 文件只修改了“distributionUrl=https\://services.gradle.org/distributions/gradle-2.5-all.zip”最后的版本号，app 目录下的 build.gradle 文件则进行了很大的修改。

先说前两个改动较小的文件。前者是因为目前的 NDK 需要一个叫“experimental”的插件。后者是因为目前的 NDK 只支持 gradle2.5，版本高了或低了都不行。

app 目录下的 build.gradle 文件需要修改的内容很多，因此笔者在上述代码中做了一些注释。在这些改动中比较重要的是 apply plugin: 'com.android.model.application'修改，以及增加了 android.ndk { moduleName = "lb" }模块。在练习时最好是除了系统版本等属性外，其他的完全按照上述改动来配置。

当修改完这 3 个跟 gradle 有关的配置文件后，Android Studio 会给出提示，如图 14-2 所示。

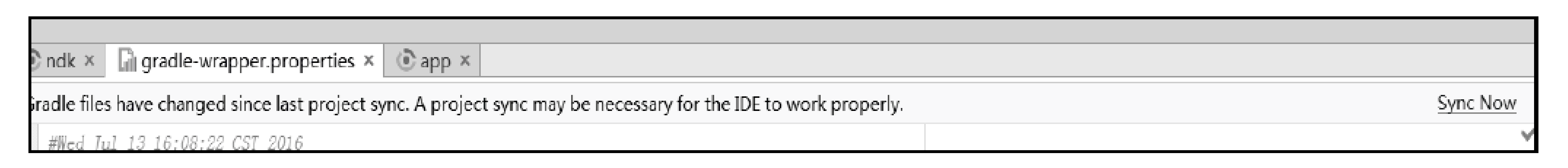


图 14-2　修改 gradle 配置之后 Studio 发出的提示

按照 Studio 的提示，点击“Sync Now”。这时如果 gradle 默认版本是 2.5，就不会有提示，NDK 开发环境就完成了。如果不是 2.5 版本，就会出现如图 14-3 所示的提示。

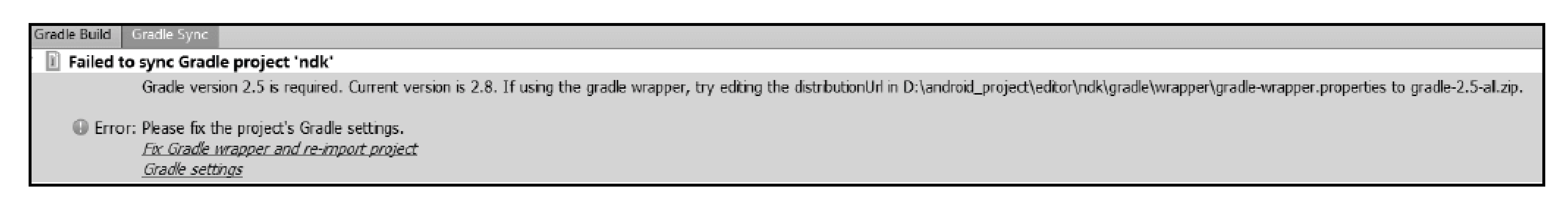


图 14-3　gradle 的版本不是 2.5 时 Studio 发出的提示

显示，它要求 2.5 版本的 gradle，但是当前的 gradle 是 2.8 的，需要更换。继续按照提示打开 Gradle Settings，选择 2.5 版本的 gradle 就可以了，如图 14-4 所示。

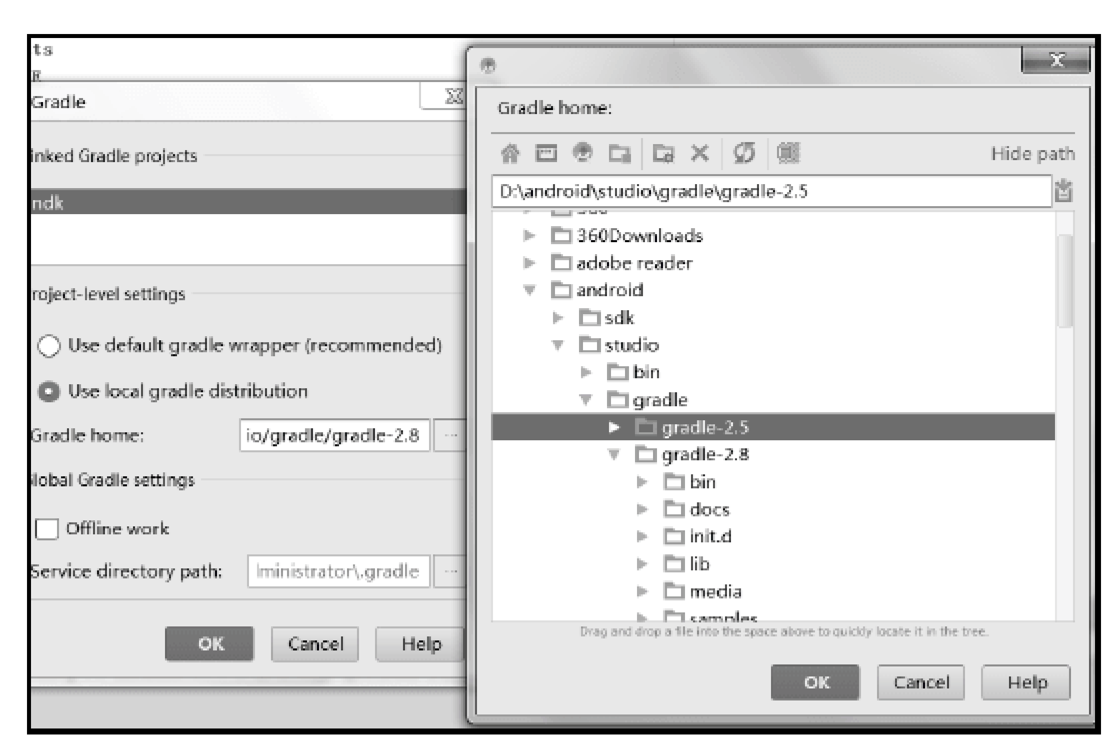


图 14-4　更改 gradle 版本为 2.5 版本

这里面的 gradle-2.5 是提前下载的 2.5 版本。下载地址是 https://gradle.org/gradle-download/，完成后解压放入“Android Studio 安装路径\gradle\”目录下即可。

至此，Android NDK 的开发环境就完成了。下面将带领大家一起开发第一个 Android NDK 应用。

14.2.2　第一个 NDK 应用

1. 创建 C/C++源文件

在 main 文件夹下建立一个 jni 文件夹，直接右击，选择“New”→“Folder”→“JNI Folder”，如图 14-5 所示。

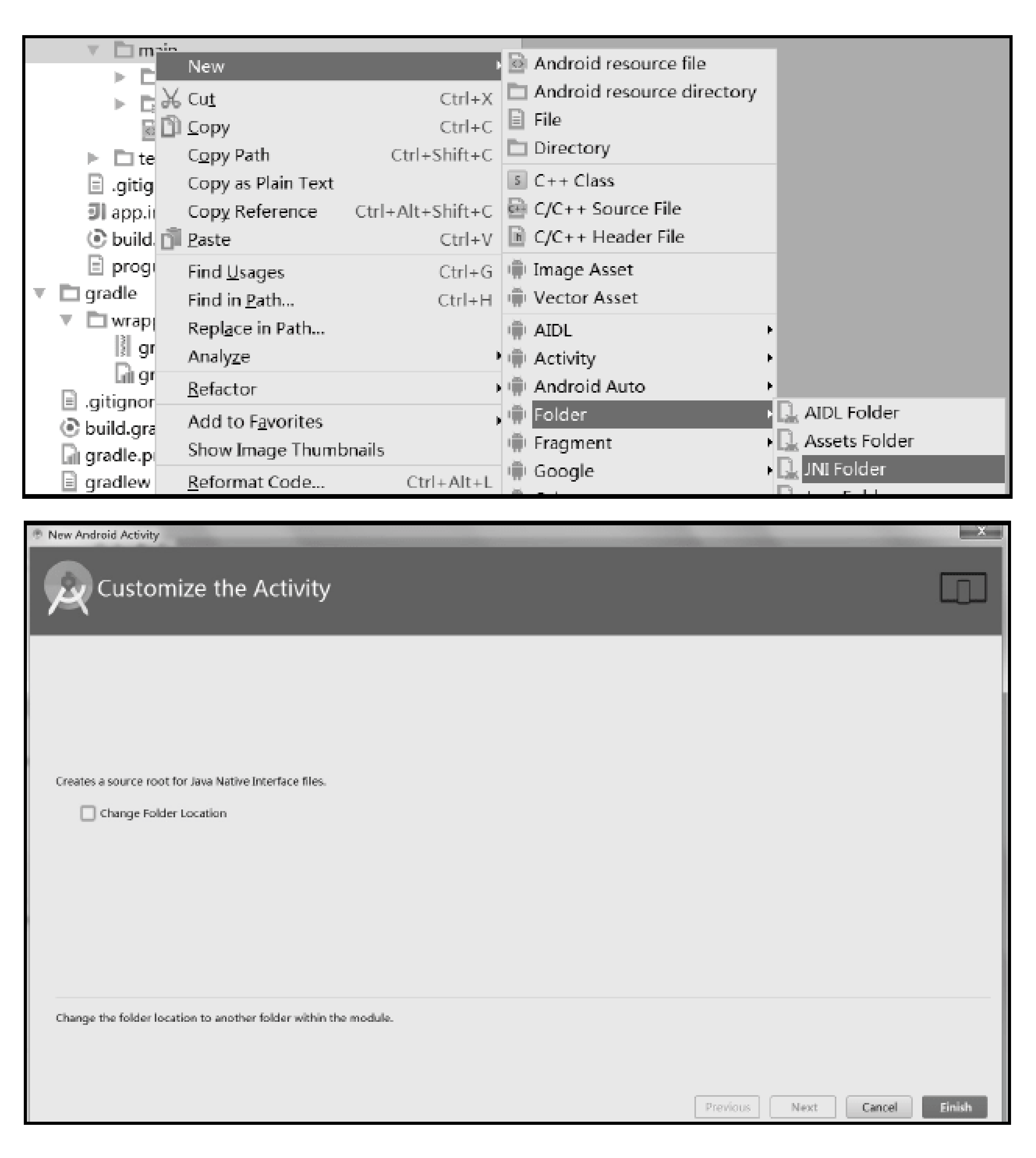


图 14-5　创建 jni 文件夹

直接点击“Finish”，不要做任何修改，这样 jni 文件夹就出现在 main 文件夹下了。接下来在 jni 文件夹内创建 C 或者 C++的源文件。打开 jni 文件夹，直接右击，选择“New”→“C/++ Source File”，如图 14-6 所示。

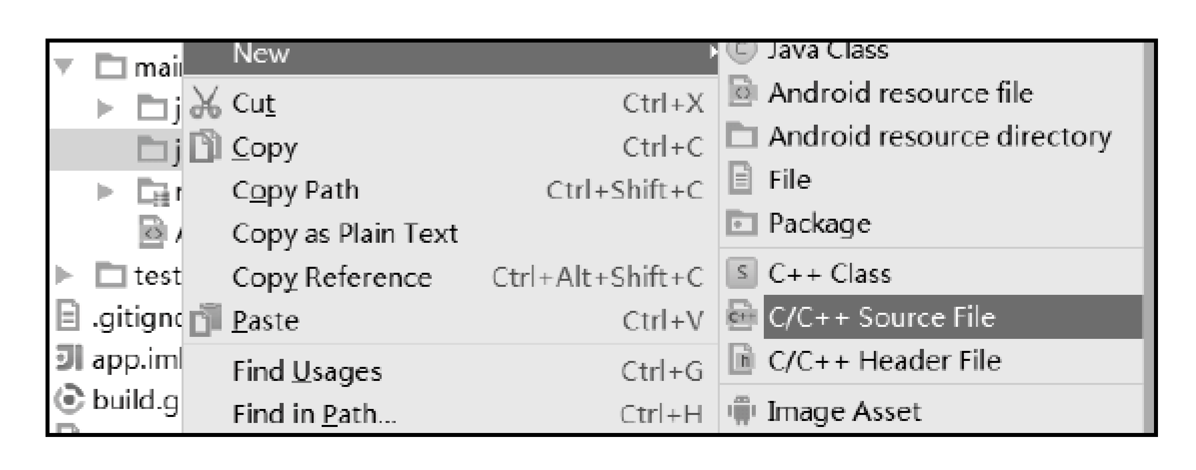


图 14-6　创建 C/C++资源文件

完成上述操作后，会出现如图 14-7 所示的界面。

按照图 14-7 所示进行选择，输入文件名，不创建头文件，点击“OK”按钮即可。此时，一个 C 或者 C++的源文件就创建完成了。这里现在不需要做任何修改，之后再做处理。

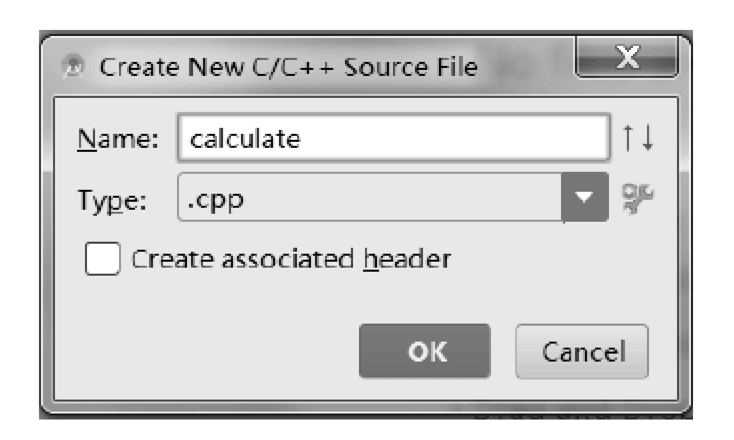


图 14-7 输入 C/C++资源文件的文件名

2. 创建 java 类以加载 C/C++库以及生产头文件

在 java 代码目录下的“com.buaa.ndk”文件夹下新建一个 LoadC 类，也可以先新建一个包再新建类，并不是必须在此文件夹下。

LoadC 类的代码如下：

```
package com.buaa.ndk;
public class LoadC {
        static {
                System.loadLibrary("lb");
        }
        public native String hello(String name);
}
```

此处代码很简单，一个静态块加载类库，类库名称为“lb”，一定不能写错，这个名称和之前在 gradle 文件中的名称要保持一致。

完成 java 类的创建后，需要“Make Project”。此时会发现在 app\build\intermediates\classes\debug 文件夹下出现了 java 类对应的.class 文件。

打开 Android Studio 上的 terminal，运行“cd app\build\intermediates\classes\debug”命令进入.class 文件的包下：

```
D:\android_project\editor\ndk>cd app\build\intermediates\classes\debug

D:\android_project\editor\ndk\app\build\intermediates\classes\debug>
```

再运行“javah -classpath . -jni com.buaa.ndk.LoadC”命令生产头文件：

```
D:\android_project\editor\ndk\app\build\intermediates\classes\debug>javah -classpath . -jni com.buaa.ndk.LoadC

D:\android_project\editor\ndk\app\build\intermediates\classes\debug>
```

读者注意，不要修改此处的命令格式，有些书籍中使用“javah -jni class 文件路径”，这样是不能成功的，必须使用“javah -classpath . -jni class 文件路径”。

此时查看 class 文件下的目录，会发现多了一个 com_buaa_ndk_LoadC.h 头文件。将此文件剪切到 jni 文件夹下。打开 com_buaa_ndk_LoadC.h 文件，代码如下：

```
/* DO NOT EDIT THIS FILE - it is machine generated */
#include <jni.h>
/* Header for class com_buaa_ndk_LoadC */

#ifndef _Included_com_buaa_ndk_LoadC
```

```
#define _Included_com_buaa_ndk_LoadC
#ifdef __cplusplus
extern "C" {
#endif
/*
 * Class:      com_buaa_ndk_LoadC
 * Method:     hello
 * Signature: (Ljava/lang/String;)Ljava/lang/String;
 */
JNIEXPORT jstring JNICALL Java_com_buaa_ndk_LoadC_hello
  (JNIEnv *, jobject, jstring);

#ifdef __cplusplus
}
#endif
#endif
```

这里 C 的语法不是我们关注的重点，下面这段代码比较特别，是读者应该认真分析的：

```
JNIEXPORT jstring JNICALL Java_com_buaa_ndk_LoadC_hello
  (JNIEnv *, jobject, jstring);
```

它声明了方法“hello”并指定了可以调用它的 Java 包是“com.buaa.ndk”、Java 类是“LoadC”。

3. 编辑源文件

编辑 calculate.cpp 这个源文件的步骤很简单，这里不做讲解，直接将代码展示如下：

```
#include "com_buaa_ndk_LoadC.h"
#include <stdio.h>
#include <string.h>

JNIEXPORT jstring JNICALL Java_com_buaa_ndk_LoadC_hello
        (JNIEnv *env, jobject obj, jstring prompt) {

    const char *str;
    str = env->GetStringUTFChars(prompt, false);
    env->ReleaseStringUTFChars(prompt, str);

    char result[80];
    strcpy(result, "hello ");
    strcat(result, str);
    puts(result);

    jstring rtstr = env->NewStringUTF(result);
    return rtstr;
}
```

这里传入一个字符串参数，返回“hello”加上此字符串。

4. 调用库方法并测试

在应用中输入一个字符串，并在 MainActivity 中获取该字符串，调用库方法，返回“hello”加上该字符串，代码如下：

```
package com.buaa.ndk;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {
    private LoadC load = new LoadC();
    private Button button;
    private TextView tv;
    private EditText et;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        button = (Button) findViewById(R.id.but);
        tv = (TextView) findViewById(R.id.result);
        et = (EditText) findViewById(R.id.edit);

        button.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                tv.setText(hello(et.getText().toString()));
                tv.setVisibility(View.VISIBLE);
            }
        });
    }

    private String hello(String name) {
        return load.hello(name);
    }
}
```

布局文件如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
```

```
        android:layout_height="match_parent"
        android:orientation="vertical">

        <EditText
            android:id="@+id/edit"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:textSize="28sp"/>

        <TextView
            android:id="@+id/result"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:visibility="gone"
            android:textSize="28sp"/>

        <Button
            android:id="@+id/but"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="点击" />
    </LinearLayout>
```

运行应用，初始界面如图 14-8 所示。输入文本，点击“点击”按钮，效果如图 14-9 所示。

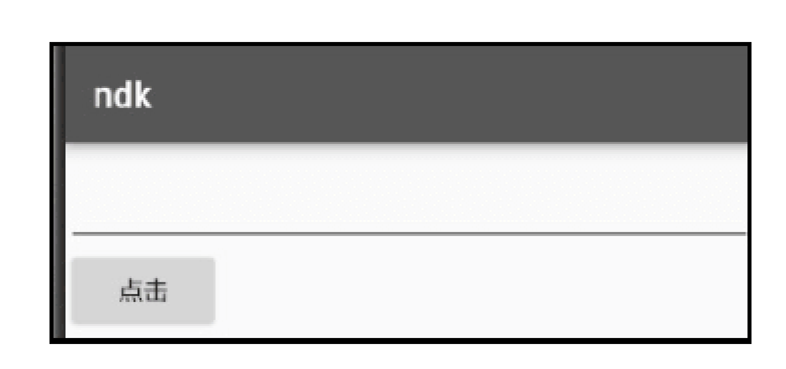


图 14-8　运行第一个 NDK 程序

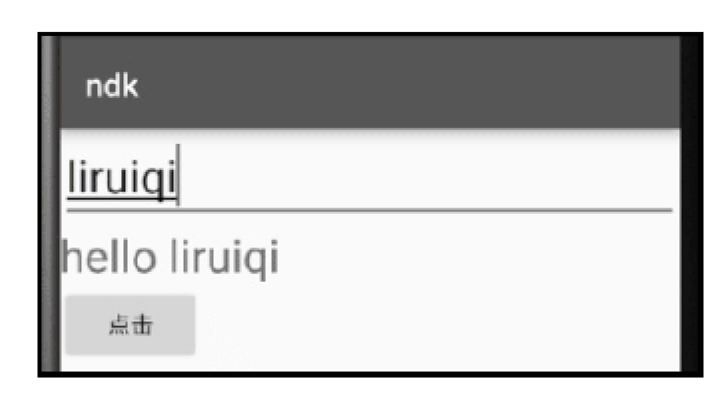
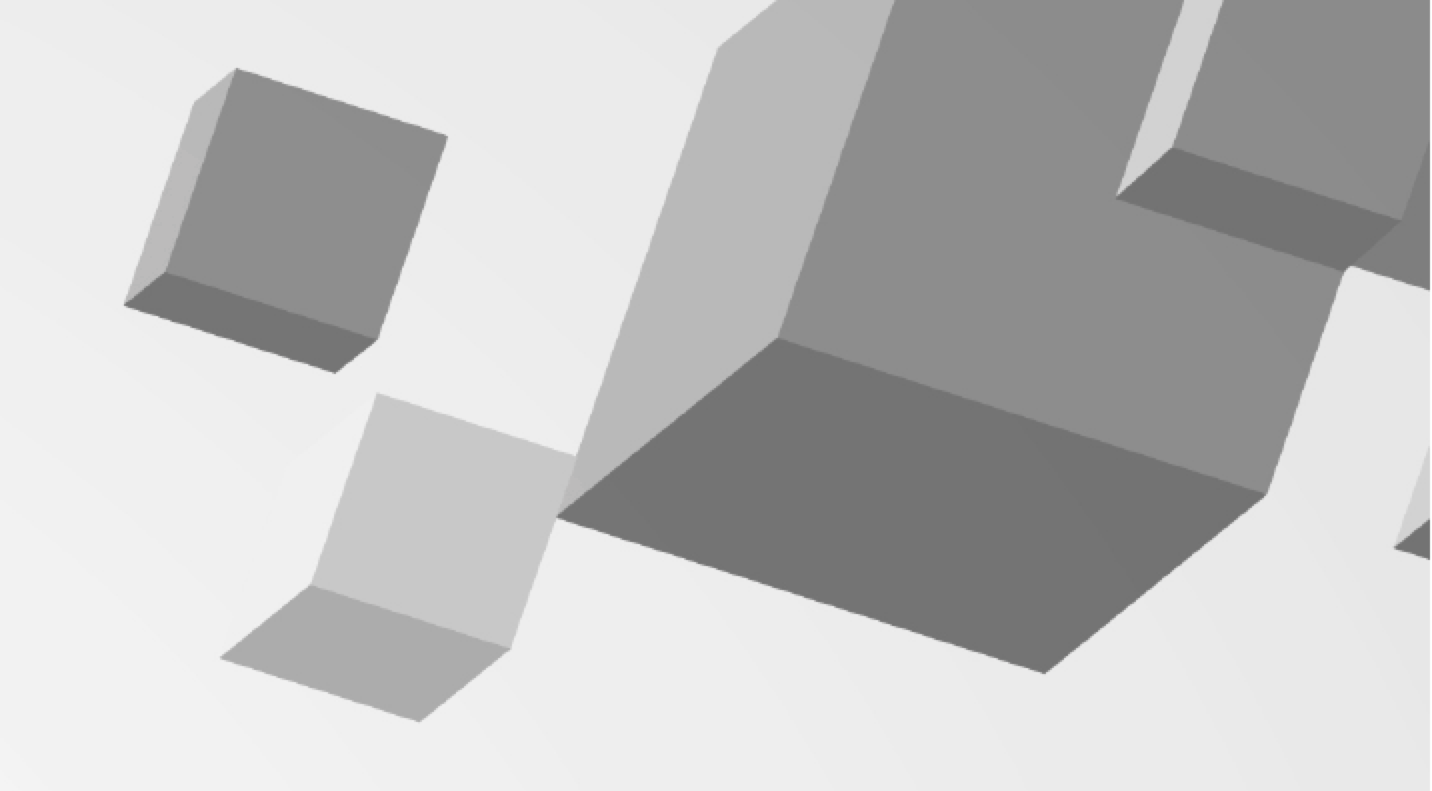


图 14-9　触发 jni 方法

这说明我们对本地库方法的调用是成功，第一个 NDK 程序也是成功的。当然，本节中的例子比较简单，不过一个 NDK 工程该有的部分全都包括在内了。如果遇到复杂状况，只需要在 C/C++源文件中开发更多的 C/C++函数，生成头文件后在 LoadC 类中调用这些函数，其他部分依旧一样。

14.3　小　　结

本章讲述了 Android NDK 开发的背景以及优势，并详细讲解了如何使用 Android Studio 进行 Android NDK 开发。关于更多的 Android NDK 开发的知识，在本章中没有进一步阐述，这是因为即使更加复杂的 Android NDK 开发，流程也是相同的，只是调用的 jni 方法更多、C/C++函数更多而已。读者如有兴趣，可以阅读 jni 相关书籍。



第 15 章 完成并发布一个产品

经过之前章节的学习，读者应该掌握了 Android 开发中常用的技术，现在我们就用这些技术开发一个完整的应用，并将之发布到应用市场上。本章将开发一个日记本，并取一个好听的名字“口袋日记”，然后将其发布到小米应用市场。

15.1 功能需求分析

一个产品的开发除了编写代码之外，更重要的是编写代码之前的功能需求分析和产品原型设计。经过对“口袋日记”的仔细分析，会发现它应该具备两个核心功能（记录功能与查看功能）与两个非核心功能（用户验证登录功能与个人展示功能）。

下面我们来详细分析为什么需要这些功能以及这些功能具体需要实现哪些内容。

（1）记录功能

作为一个日记本，记录是它最为直接的需求。如果日记本不能记录内容，就不能被称作日记本。传统式日记本都是通过文字记录生活中的点点滴滴，在 Android 应用中，我们还可以使用图片、音频、视频的方式来实现记录功能。同时，为了方便用户之后的查看，日记中需要记录时间和地点，如果让用户手动填写，用户体验可能会不太好，因此记录功能中还需要帮助用户获取记录时间和地点。最终，这些记录内容将保存在本地应用中。

（2）查看功能

使用“口袋日记”记录好一天各种幸福的或者悲伤的事情之后，突然有一天想再看看当时的心情就需要用到查看功能了。具体来说，只需要一个列表就能够展示所有的日记，并可以通过点击具体的条目来展现该条日记。

（3）用户验证登录功能

可能有些读者会认为这个是多余的，因为这毕竟是一个单机应用，不需要服务器交互。但是，日记本是一个非常隐私的应用，因此需要设计一个用户验证登录功能，用以保证其他人不能轻易观看到用户的日记。

（4）个人展示功能

个人展示功能并不是核心功能点，可以说是可有可无的，甚至可以不加此功能点。但是加入了之后，还可以让用户填写个人说明等，比如填写较为励志的说明，若用户经常看到则可能会起到无形的激励作用。从应用的完整度上来说，加入此功能也较有好处。

可能有的读者会觉得功能较少，只有 4 个，其实这里面将涉及本书之前所学习的很多知识，并且需要具备综合运用这些知识的能力。而且本章主要是想让读者了解一个产品从开发到发布再到应用市场的整个过程，所以在产品的选择上并没有选择十分复杂的。另外，在实际的开发实践中，还需要有产品原型设计这样一个步骤，读者也需要知道，但这里略去。下面我们就开始进行产品的开发工作。

15.2　功能开发（上）

15.2.1　程序概览

在讲解开发之前，我们从整体上演示一下代码结构（见图 15-1）。

读者可能会发现，在 Android Studio 的“com.buaa.diary”包下新建了几个包。其中，activity 包用以存放 activity 类的代码，adapter 包用于存放各种适配器代码，database 包用于存放数据库相关代码，entity 包用于存放 java 实体类，fragment 包用于存放 fragment 类代码，util 包用于存放一些工具类。res 文件夹下的一些文件夹并不需要修改。

```
app
  build
  libs
  src
    androidTest
    main
      java
        com.buaa.diary
          activity
          adapter
          database
          entity
          fragment
          util
      res
        drawable
        layout
        mipmap-hdpi
        mipmap-mdpi
        mipmap-xhdpi
        mipmap-xxhdpi
        mipmap-xxxhdpi
        values
        values-w820dp
      AndroidManifest.xml
    test
  .gitignore
  app.iml
  build.gradle
  proguard-rules.pro
```

图 15-1　项目整体的代码结构

15.2.2　数据库设计与开发

通过前面的分析，我们发现本应用只需要一个日记信息表即可。该表的字段有 id、标题、作者、日期、地址、保存内容的路径、内容。创建表的 SQL 语句如下：

```
"CREATE TABLE IF NOT EXISTS " +
        "diary (diary_id INTEGER primary key autoincrement," +
        "title varchar(64),author varchar(64),date varchar(64)," +
        "address varchar(64),uri varchar(256),content varchar(1024))"
```

我们在 database 包下创建一个 OpenHelper 类，创建表和升级表，代码如下：

```
package com.buaa.diary.database;

import android.content.Context;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;

public class OpenHelper extends SQLiteOpenHelper {

    private static final String name = "diary.db";
    private static final int version = 1;

    public OpenHelper(Context context) {
        super(context, name, null, version);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        db.execSQL("CREATE TABLE IF NOT EXISTS " +
```

```
                    "diary (diary_id INTEGER primary key autoincrement," +
                    "title varchar(64),author varchar(64),date varchar(64)," +
                    "address varchar(64),uri varchar(256),content varchar(1024))");
        }
        @Override
        public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
            if (newVersion > oldVersion) {
                //修改表，暂时可以不用
            }
        }
    }
```

需要说明，这里只是做演示使用，所以没有做数据库版本升级的处理。然后我们在 entity 包下创建一个 Diary 类来作为该表对应的实体类，代码如下：

```
package com.buaa.diary.entity;

import java.io.Serializable;

public class Diary implements Serializable {
    private int diaryId;
    private String author;
    private String title;
    private String date;
    private String address;
    private String uri;
    private String content;

    public int getDiaryId() {
        return diaryId;
    }

    public void setDiaryId(int diaryId) {
        this.diaryId = diaryId;
    }

    public String getAuthor() {
        return author;
    }

    public void setAuthor(String author) {
        this.author = author;
    }

    public String getTitle() {
        return title;
```

```
        }
        public void setTitle(String title) {
            this.title = title;
        }
        public String getAddress() {
            return address;
        }
        public void setAddress(String address) {
            this.address = address;
        }
        public String getDate() {
            return date;
        }
        public void setDate(String date) {
            this.date = date;
        }
        public String getUri() {
            return uri;
        }
        public void setUri(String uri) {
            this.uri = uri;
        }
        public String getContent() {
            return content;
        }
        public void setContent(String content) {
            this.content = content;
        }
    }
```

同时，在 database 包下创建一个真正完成对 diary 表进行增、删、改、查操作的 DiaryDao 类，代码如下：

```
package com.buaa.diary.database;

import android.content.ContentValues;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;

import com.buaa.diary.entity.Diary;
```

```
import java.util.ArrayList;
import java.util.List;

public class DiaryDao {
    private SQLiteDatabase db;

    public DiaryDao(SQLiteDatabase sqLiteDatabase) {
        this.db = sqLiteDatabase;
    }

    public boolean insert(Diary diary) {
        ContentValues contentValues = new ContentValues();
        contentValues.put("author", diary.getAuthor());
        contentValues.put("title", diary.getTitle());
        contentValues.put("date", diary.getDate());
        contentValues.put("address", diary.getAddress());
        contentValues.put("uri", diary.getUri());
        contentValues.put("content", diary.getContent());
        long insertResult = db.insert("diary", null, contentValues);
        if (insertResult == -1) {
            return false;
        }
        return true;
    }

    public boolean delete(Diary diary) {
        if (diary == null) {
            db.delete("diary", "", null);
            return true;
        }
        int deleteResult = db.delete("diary", "diary_id=?",
                new String[]{diary.getDiaryId() + ""});
        if (deleteResult == 0) {
            return false;
        }
        return true;
    }

    public boolean update(Diary diary) {
        ContentValues contentValues = new ContentValues();
        contentValues.put("author", diary.getAuthor());
        contentValues.put("title", diary.getTitle());
```

```
            contentValues.put("date", diary.getDate());
            contentValues.put("address", diary.getAddress());
            contentValues.put("uri", diary.getUri());
            contentValues.put("content", diary.getContent());
            int updateResult = db.update("diary", contentValues, "diary_id=?",
                    new String[]{diary.getDiaryId() + ""});
            if (updateResult == 0) {
                return false;
            }
            return true;
        }

        public List<Diary> queryAll() {
            List<Diary> diaryList = new ArrayList<>();
            Cursor cursor = db.query("diary", null, null,
                    null, null, null, null);
            while (cursor.moveToNext()) {
                Diary diary = new Diary();
                diary.setDiaryId(cursor.getInt(0));
                diary.setTitle(cursor.getString(1));
                diary.setAuthor(cursor.getString(2));
                diary.setDate(cursor.getString(3));
                diary.setAddress(cursor.getString(4));
                diary.setUri(cursor.getString(5));
                diary.setContent(cursor.getString(6));
                diaryList.add(diary);
            }
            return diaryList;
        }
    }
```

到此，应用的数据库部分开发已经完成。由于这里使用到的所有知识之前都有讲解，因此此处不再赘述。

完成了数据库的相关开发之后，我们来实现 15.1 节所说的 4 个功能，这里先实现用户登录验证功能。在此声明，本章开发的应用可能在产品设计上存在一些不合理之处，我们的重点是学习如何开发一个完整应用并将其发布到应用市场。

15.2.3　用户登录验证

要想实现用户登录验证，我们需要在 activity 包下创建一个新的 Activity 类 LoginActivity，并修改布局文件 activity_login.xml 如下：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
```

```
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context="com.buaa.diary.activity.LoginActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:layout_marginTop="60dp"
        android:text="口袋日记"
        android:textSize="22sp" />

    <EditText
        android:id="@+id/username"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="20dp"
        android:hint="请输入用户名"
        android:textSize="22sp" />

    <EditText
        android:id="@+id/password"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="20dp"
        android:hint="请输入密码"
        android:inputType="textPassword"
        android:textSize="22sp" />

    <Button
        android:id="@+id/login"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="20dp"
        android:tag="101"
        android:text="进入日记本"
        android:textColor="@color/colorAccent"
        android:textSize="22sp" />

</LinearLayout>
```

这里的代码很简单，只包括两个用于输入用户名和密码的 EditText 和一个用户触发事件

的 Button，还有一个展示文字的 TextView，比较容易理解。下面修改 LoginActivity 类，这里有点复杂，代码如下：

```
package com.buaa.diary.activity;

import android.app.AlertDialog;
import android.content.DialogInterface;
import android.content.Intent;
import android.content.SharedPreferences;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.text.InputType;
import android.view.Gravity;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.LinearLayout;
import android.widget.TextView;
import android.widget.Toast;

import com.buaa.diary.R;
import com.buaa.diary.util.Configure;
import com.buaa.diary.util.Util;

public class LoginActivity extends AppCompatActivity implements View.OnClickListener {
    private EditText nameEditText;
    private EditText passwordEditText;
    private Button loginButton;
    private AlertDialog registerDialog;
    private EditText registerName;
    private EditText registerPassword;
    private SharedPreferences sharedPreferences;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_login);
        initView();
        initData();
    }

    private void initView() {
        nameEditText = (EditText) findViewById(R.id.username);
```

```
        passwordEditText = (EditText) findViewById(R.id.password);
        loginButton = (Button) findViewById(R.id.login);
        loginButton.setTag(101);
        loginButton.setOnClickListener(this);
    }

    private void initData() {
        sharedPreferences = getSharedPreferences("user", MODE_PRIVATE);
        String name = sharedPreferences.getString("name", "");
        String address = sharedPreferences.getString("password", "");
        if (name.equals("") && address.equals("")) {
            register();
        } else {
            nameEditText.setText(name);
        }
    }

    private void register() {
        registerDialog = new Util().getDialog(this, registerDialogView());
        registerDialog.show();

    }

    private View registerDialogView() {
        LinearLayout linearLayout = new LinearLayout(this);
        linearLayout.setOrientation(LinearLayout.VERTICAL);
        TextView titleText = new TextView(this);
        titleText.setText("设置账号");
        titleText.setTextSize(20);
        titleText.setGravity(Gravity.CENTER);
        titleText.setPadding(0, 20, 0, 10);
        registerName = new EditText(this);
        registerName.setHint("请输入用户名");
        registerPassword = new EditText(this);
        registerPassword.setInputType(InputType.TYPE_TEXT_VARIATION_PASSWORD);
        registerPassword.setHint("请输入密码");

        LinearLayout buttonLinearLayout = new LinearLayout(this);
        buttonLinearLayout.setOrientation(LinearLayout.HORIZONTAL);
        buttonLinearLayout.setGravity(Gravity.CENTER);

        Button registerButton = new Button(this);
        registerButton.setText("注册");
```

```
            registerButton.setOnClickListener(this);
            registerButton.setTag(102);
            Button cancelButton = new Button(this);
            cancelButton.setText("取消");
            cancelButton.setOnClickListener(this);
            cancelButton.setTag(103);

            buttonLinearLayout.addView(registerButton);
            buttonLinearLayout.addView(cancelButton);

            linearLayout.addView(titleText);
            linearLayout.addView(registerName);
            linearLayout.addView(registerPassword);
            linearLayout.addView(buttonLinearLayout);
            return linearLayout;
        }

        private void alertRegister(final String name, final String password) {
            AlertDialog.Builder builder = new AlertDialog.Builder(this);
            builder.setCancelable(false);
            builder.setMessage("您正在设置账号，这个账号将保护您的隐私不被其他人看到。" +
                    "现在请点击完成结束设置，或点击取消重新设置");
            builder.setPositiveButton("完成", new DialogInterface.OnClickListener() {
                @Override
                public void onClick(DialogInterface dialog, int which) {
                    SharedPreferences.Editor editor = sharedPreferences.edit();
                    editor.putString("name", name);
                    editor.putString("password", password);
                    editor.commit();
                    registerDialog.dismiss();
                    nameEditText.setText(name);
                }
            });
            builder.setNegativeButton("取消", null);
            builder.show();
        }

        private void alertLoginError() {
            AlertDialog.Builder builder = new AlertDialog.Builder(this);
            builder.setCancelable(false);
            builder.setMessage("您输入的用户名、密码验证不成功，是否需要设置用户名、密码？");
            builder.setPositiveButton("确定", new DialogInterface.OnClickListener() {
                @Override
```

```
                public void onClick(DialogInterface dialog, int which) {
                    register();
                }
            });
            builder.setNegativeButton("取消", null);
            builder.show();
        }

        @Override
        public void onClick(View v) {
            switch ((Integer) v.getTag()) {
                case 101:
                    String loginName = nameEditText.getText().toString();
                    String loginPassword = passwordEditText.getText().toString();
                    if ("".equals(loginName) || "".equals(loginPassword)) {
                        alertLoginError();
                        break;
                    } else if (sharedPreferences.getString("name", "").equals(loginName)
                            && sharedPreferences.getString("password", "").equals(loginPassword)) {
                        Toast.makeText(this,
                                "验证成功!",
                                Toast.LENGTH_SHORT).show();
                        Intent intent = new Intent(this, MainActivity.class);
                        intent.putExtra("from", Configure.FROM_LOGIN_ACTIVITY);
                        startActivity(intent);
                        this.finish();
                        break;
                    } else {
                        alertLoginError();
                    }
                    break;
                case 102:
                    String name = registerName.getText().toString();
                    String password = registerPassword.getText().toString();
                    if ("".equals(name) || "".equals(password)) {
                        Toast.makeText(this,
                                "输入的值不能为空！",
                                Toast.LENGTH_LONG).show();
                        break;
                    } else {
                        alertRegister(name, password);
                    }
                    break;
```

```
                case 103:
                    Toast.makeText(this,
                            "您取消了设置账号，这有可能导致您的日记隐私处于不安全状态！",
                            Toast.LENGTH_LONG).show();
                    registerDialog.dismiss();
                    break;
            }
        }
    }
```

这里的代码比较多，所以先对其中的逻辑进行讲解。这里先通过 initView()方法对 View 中的各个控件进行初始化，然后用 initData()方法进行数据初始化。在 initData()方法中判断 ShardPreferences 中有无保存用户信息，如果没有就让用户进行注册，这里的注册使用的是一个 Dialog；如果有就让用户登录，登录正确则进入 MainActivity（日记读写的类），不正确则提示让用户注册。至于代码，经过之前的学习，读者应该已经熟悉，这里不再讲解。

15.2.4 工具类

在 15.2.3 小节中，我们使用了 Configure 类和 Util 类的内容，它们都在 util 包下。这两个类都是工具类，Configure 类主要用于存放一些常量，在 Util 类中则主要是一些常用的方法。Configure 类的代码如下：

```
package com.buaa.diary.util;

public class Configure {
    public static final int LOCATION_PERMISSION_CODE = 1001;
    public static final int LOCATION_MESSAGE_CODE = 1002;
    public static final int IMAGE_CAMERA = 1003;
    public static final int IMAGE_ALBUM = 1004;
    public static final int IMAGE_PERMISSION_CODE = 1005;
    public static final int CAMERA_PERMISSION_CODE = 1006;
    public static final int FROM_SHOW_DIARY_ACTIVITY = 1007;
    public static final int FROM_LOGIN_ACTIVITY = 1008;
}
```

读者会发现，这里的常量并不仅是在 LoginActivity 中使用的那些，还有一些是被其他类使用的，这在之后会讲解。Util 类代码如下：

```
package com.buaa.diary.util;

import android.app.AlertDialog;
import android.content.Context;
import android.content.pm.PackageManager;
import android.os.Build;
import android.support.v4.app.ActivityCompat;
```

```
import android.support.v7.app.AppCompatActivity;
import android.view.View;

public class Util {

    public boolean checkPermission(String[] permissions, AppCompatActivity activity) {
        boolean flag = false;
        if (Build.VERSION.SDK_INT >= 23) {
            int permission_granted = PackageManager.PERMISSION_GRANTED;
            for (int i = 0; i < permissions.length; i++) {
                int checkPermission = ActivityCompat.checkSelfPermission
                        (activity, permissions[i]);
                if (permission_granted != checkPermission) {
                    flag = true;
                    break;
                }
            }
        }
        return flag;
    }

    public AlertDialog getDialog(Context context, View view) {
        AlertDialog.Builder builder = new AlertDialog.Builder(context);
        builder.setCancelable(false);
        builder.setView(view);
        return builder.create();
    }
}
```

Util 类和 Configure 类一样，代码都很容易理解。这里的 Util 类中封装了一个获取 Dialog 的方法和一个检查权限的方法。经过之前章节的学习，读者对这些应该都很熟悉了，这里不再讲解。另外，由于在接下来进行日记的读写开发时需要获得用户的位置，因此需要对用户进行定位。这里在 util 包下新建一个 LocationUtil 类，用于对 Location 进行处理，代码如下：

```
package com.buaa.diary.util;

import android.Manifest;
import android.content.Context;
import android.location.Address;
import android.location.Geocoder;
import android.location.Location;
import android.location.LocationListener;
import android.location.LocationManager;
import android.os.Bundle;
```

```
import android.os.Handler;
import android.os.Message;
import android.support.v4.app.Fragment;
import android.support.v7.app.AppCompatActivity;
import android.widget.Toast;

import java.io.IOException;
import java.util.List;

public class LocationUtil implements LocationListener {
    private Util util;
    private LocationManager locationManager;
    private AppCompatActivity activity;
    private Handler handler;
    private String[] locationPermissions = new String[]{
            Manifest.permission.ACCESS_COARSE_LOCATION,
            Manifest.permission.ACCESS_FINE_LOCATION
    };
    private Fragment fragment;

    public LocationUtil(AppCompatActivity activity, Handler handler) {
        util = new Util();
        this.activity = activity;
        this.handler = handler;
    }

    public void removeLocationUpdates() {
        if (locationManager != null) {
            boolean locationPermissionFlag = util.checkPermission(locationPermissions,
                    activity);
            if (!locationPermissionFlag) {
                locationManager.removeUpdates(this);
                locationManager = null;
            }
        }
    }

    public void getLocation(Fragment fragment) {
        this.fragment = fragment;
        boolean locationPermissionFlag = util.checkPermission(locationPermissions,
                activity);
        if (locationPermissionFlag) {
            fragment.requestPermissions(locationPermissions,
```

```
Configure.LOCATION_PERMISSION_CODE);
            }else{
                locationManager = (LocationManager) activity.getSystemService(Context.
                        LOCATION_SERVICE);
                String provider;
                List<String> providerList = locationManager.getProviders(true);
                if (providerList.contains(LocationManager.GPS_PROVIDER)) {
                    provider = LocationManager.GPS_PROVIDER;
                } else if (providerList.contains(LocationManager.NETWORK_PROVIDER)) {
                    provider = LocationManager.NETWORK_PROVIDER;
                } else {
                    Toast.makeText(activity, "请连接网络或打开 GPS",
                            Toast.LENGTH_LONG).show();
                    return;
                }
                Location location = locationManager.getLastKnownLocation(provider);
                locationManager.requestLocationUpdates(provider, 2000, 10, this);
                if (location != null) {
                    getLocation(location);
                }
            }
        }

        private void getLocation(Location location) {
            Geocoder geocoder = new Geocoder(activity);
            try {
                List<Address> addresses = geocoder.getFromLocation(
                        location.getLatitude(), location.getLongitude(), 1);
                Address address = addresses.get(0);
                String result = address.getAddressLine(1) + address.getFeatureName();
                Message message = handler.obtainMessage();
                message.what = Configure.LOCATION_MESSAGE_CODE;
                message.obj = result;
                handler.sendMessage(message);
            } catch (IOException e) {
                e.printStackTrace();
            }
        }

        @Override
        public void onLocationChanged(Location location) {
            getLocation(fragment);
        }
```

```
    @Override
    public void onStatusChanged(String provider, int status, Bundle extras) {

    }

    @Override
    public void onProviderEnabled(String provider) {

    }

    @Override
    public void onProviderDisabled(String provider) {

    }
}
```

本类与本书地理信息技术一章中讲解的内容大体相同。

15.3 功能开发（下）

当用户经过登录验证且正确时进入 MainActivity。我们在 MainActivity 中将进行日记的读写功能开发，具体来说就是使用 ViewPager 加 3 个 Fragment。这 3 个 Fragment 分别对应日记记录、日记查询、个人中心 3 个部分。

15.3.1 日记记录

正如前文所说，记录功能和其他功能都是使用的 Fragment，宿主正是 MainActivity，所以这里先在 activity 包下创建一个 MainActivity 类，并修改布局文件 activity_main.xml 如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".activity.MainActivity">


    <android.support.v4.view.ViewPager
        android:id="@+id/viewPager"
```

```
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1"></android.support.v4.view.ViewPager>

    <View
        android:layout_width="match_parent"
        android:layout_height="1px"
        android:background="@color/colorAccent"></View>

    <android.support.design.widget.TabLayout
        android:id="@+id/tab"
        android:layout_width="match_parent"
        android:layout_height="60dp"
        app:tabMode="fixed"></android.support.design.widget.TabLayout>

</LinearLayout>
```

这里使用了一个 ViewPager 和一个 TabLayout。需要说明的是，使用 TabLayout 需要在 build.gradle 文件中修改 dependencies 模块：

```
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    testCompile 'junit:junit:4.12'
    compile 'com.android.support:support-v4:23.4.0'
    compile 'com.android.support:design:23.4.0'
}
```

这里加入的是“compile 'com.android.support:design:23.4.0'”。当然，这里面的版本需要根据开发时使用的 SDK 版本而定，因为笔者使用的 SDK 版本是 23.4.0，所以这里的包名后面的后缀就变成了 23.4.0。

修改 MainActivity 如下：

```
package com.buaa.diary.activity;

import android.support.design.widget.TabLayout;
import android.support.v4.app.Fragment;
import android.support.v4.app.FragmentManager;
import android.support.v4.view.ViewPager;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.view.KeyEvent;
import android.widget.Toast;

import com.buaa.diary.R;
import com.buaa.diary.adapter.FragmentAdapter;
```

```
import com.buaa.diary.fragment.ListDiaryFragment;
import com.buaa.diary.fragment.PersonFragment;
import com.buaa.diary.fragment.WriteDiaryFragment;
import com.buaa.diary.util.Configure;

import java.util.ArrayList;
import java.util.List;

public class MainActivity extends AppCompatActivity {
    private long exitTime = 0;
    private int fromWhich = 0;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        initData();
        initViewForFragment();
    }

    private void initData() {
        fromWhich = getIntent().getIntExtra("from", 0);
    }

    private void initViewForFragment() {
        FragmentManager fragmentManager = getSupportFragmentManager();
        List<Fragment> fragments = new ArrayList<>();
        fragments.add(WriteDiaryFragment.newInstance(this));
        fragments.add(ListDiaryFragment.newInstance(this));
        fragments.add(PersonFragment.newInstance(this));

        List<String> tabs = new ArrayList<>();
        tabs.add("写日记");
        tabs.add("日记中心");
        tabs.add("个人中心");
        FragmentAdapter fragmentAdapter = new FragmentAdapter(fragmentManager, fragments, tabs);

        ViewPager viewPager = (ViewPager) findViewById(R.id.viewPager);
        viewPager.setAdapter(fragmentAdapter);
        TabLayout tabLayout = (TabLayout) findViewById(R.id.tab);
        tabLayout.setupWithViewPager(viewPager);
        if (fromWhich == Configure.FROM_SHOW_DIARY_ACTIVITY) {
            tabLayout.getTabAt(1).select();
```

```
            }
        }

        @Override
        public boolean onKeyDown(int keyCode, KeyEvent event) {
            if (keyCode == KeyEvent.KEYCODE_BACK) {
                if (System.currentTimeMillis() - exitTime > 2000) {
                    exitTime = System.currentTimeMillis();
                    Toast.makeText(this, "再按退出", Toast.LENGTH_SHORT).show();
                } else {
                    finish();
                    System.exit(0);
                }
            }
            return true;
        }
    }
```

是不是很熟悉？在本书讲解基本控件时 ViewPager、Fragment、TabLayout 三者结合使用，与上述内容基本相同。另外，在 MainActivity 中，我们设置了按手机“回退键”的处理方式，即第一次按会提示“再按退出”，两秒内再按就会退出程序。另外，这里还需要一个适配器。在 adapter 包下新建一个 FragmentAdapter 类，作为 ViewPager 的适配器，代码如下：

```
package com.buaa.diary.adapter;

import android.support.v4.app.Fragment;
import android.support.v4.app.FragmentManager;
import android.support.v4.app.FragmentPagerAdapter;

import java.util.List;

public class FragmentAdapter extends FragmentPagerAdapter {

    private List<Fragment> fragmentList;
    private List<String> tabList;

    public FragmentAdapter(FragmentManager fm, List<Fragment> fragmentList, List<String> tabList) {
        super(fm);
        this.fragmentList = fragmentList;
        this.tabList = tabList;
    }

    @Override
    public Fragment getItem(int position) {
```

```
            return fragmentList.get(position);
        }

        @Override
        public int getCount() {
            return fragmentList.size();
        }

        @Override
        public CharSequence getPageTitle(int position) {
            return tabList.get(position);
        }
    }
```

从 MainActivity 中可以看出 WriteDiaryFragment 类是处理日记记录功能的 Fragment，布局文件 write_diary.xml 如下：

```
    <?xml version="1.0" encoding="utf-8"?>
    <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:focusable="true"
        android:focusableInTouchMode="true"
        android:orientation="vertical">

        <EditText
            android:id="@+id/title"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_marginTop="30dp"
            android:hint="请输入标题"
            android:singleLine="true"
            android:textSize="18sp" />

        <TextView
            android:id="@+id/date"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="center"
            android:textSize="16sp" />

        <TextView
            android:id="@+id/address"
            android:layout_width="wrap_content"
```

```
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:textSize="14sp" />

    <View
        android:layout_width="match_parent"
        android:layout_height="10dp" />

    <ScrollView
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1">

        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:orientation="vertical">

            <EditText
                android:id="@+id/text_content"
                android:layout_width="match_parent"
                android:layout_height="200dp"
                android:gravity="top"
                android:hint="请输入内容"
                android:textSize="16sp" />

            <LinearLayout
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:orientation="horizontal">

                <ImageView
                    android:id="@+id/image1"
                    android:layout_width="0dp"
                    android:layout_height="200dp"
                    android:layout_weight="1"
                    android:visibility="gone" />

                <ImageView
                    android:id="@+id/image2"
                    android:layout_width="0dp"
                    android:layout_height="200dp"
                    android:layout_weight="1"
```

```
                        android:visibility="gone" />
                </LinearLayout>

                <View
                    android:layout_width="match_parent"
                    android:layout_height="10dp" />

                <LinearLayout
                    android:layout_width="match_parent"
                    android:layout_height="wrap_content"
                    android:orientation="horizontal">

                    <ImageView
                        android:id="@+id/image3"
                        android:layout_width="0dp"
                        android:layout_height="200dp"
                        android:layout_weight="1"
                        android:visibility="gone" />

                    <ImageView
                        android:id="@+id/image4"
                        android:layout_width="0dp"
                        android:layout_height="200dp"
                        android:layout_weight="1"
                        android:visibility="gone" />

                </LinearLayout>
            </LinearLayout>
        </ScrollView>

        <ImageView
            android:id="@+id/add_content"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="center"
            android:src="@drawable/add" />
    </LinearLayout>
```

这里使用线性布局，最上方使用 EditText 来输入标题，接下来使用 TextView 来显示日期和地址。接着使用一个 View 来给两部分加入一些空间。然后使用一个 ScrollView 来控制输入内容，使内容可以进行滑动，由于 ScrollView 内只能加入一个控件，因此在内部嵌套一个 LinearLayout，此 LinearLayout 内先使用一个 EditText 来输入日记内容，在下面再使用隐藏的 ImageView 来加入图片，利用 ImageView 来触发加入图片和将隐藏的 ImageView 显示事件。

这里最多只能上传 4 张图片，读者可能会问为什么这样处理，而不是在代码中动态加入 ImageView，答案很简单——在代码中动态添加 ImageView 不利于控制格式。

完成布局文件之后，修改 WriteDiaryFragment 类如下：

```
package com.buaa.diary.fragment;

import android.Manifest;
import android.app.AlertDialog;
import android.content.Context;
import android.content.Intent;
import android.content.pm.PackageManager;
import android.database.sqlite.SQLiteDatabase;
import android.graphics.Bitmap;
import android.net.Uri;
import android.os.Build;
import android.os.Bundle;
import android.os.Handler;
import android.os.Message;
import android.provider.MediaStore;
import android.support.v4.app.Fragment;
import android.support.v7.app.AppCompatActivity;
import android.view.Gravity;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.Button;
import android.widget.EditText;
import android.widget.ImageView;
import android.widget.LinearLayout;
import android.widget.TextView;
import android.widget.Toast;

import com.buaa.diary.R;
import com.buaa.diary.activity.MainActivity;
import com.buaa.diary.activity.ShowDiaryActivity;
import com.buaa.diary.database.DiaryDao;
import com.buaa.diary.database.OpenHelper;
import com.buaa.diary.entity.Diary;
import com.buaa.diary.util.Configure;
import com.buaa.diary.util.LocationUtil;
import com.buaa.diary.util.Util;

import java.io.File;
```

```
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.text.SimpleDateFormat;
import java.util.Date;

public class WriteDiaryFragment extends Fragment implements View.OnClickListener {
    Handler handler = new Handler() {
        @Override
        public void handleMessage(Message msg) {
            super.handleMessage(msg);
            switch (msg.what) {
                case Configure.LOCATION_MESSAGE_CODE:
                    address.setText("于" + msg.obj);
                    break;
            }
        }
    };
    private static AppCompatActivity activity;
    private LocationUtil locationUtil;
    private TextView address;
    private TextView date;
    private ImageView addContent;
    private AlertDialog addContentDialog;
    private AlertDialog addImageDialog;
    private EditText title;
    private EditText textContent;
    private ImageView image1;
    private ImageView image2;
    private ImageView image3;
    private ImageView image4;
    private int imageCount = 0;
    private String uriList = "";
    private Diary diary;

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        View view = inflater.inflate(R.layout.write_diary, container, false);
        initView(view);
        return view;
    }
```

```
    @Override
    public void onResume() {
        super.onResume();
        initLocation();
    }

    private void initLocation() {
        locationUtil = new LocationUtil(activity, handler);
        locationUtil.getLocation(this);
    }

    public static WriteDiaryFragment newInstance(MainActivity activity) {
        WriteDiaryFragment.activity = activity;
        WriteDiaryFragment fragment = new WriteDiaryFragment();
        return fragment;
    }

    @Override
    public void onDestroy() {
        super.onDestroy();
        locationUtil.removeLocationUpdates();
        locationUtil = null;
    }

    private void initView(View view) {
        address = (TextView) view.findViewById(R.id.address);
        date = (TextView) view.findViewById(R.id.date);
        Date currentTime = new Date();
        SimpleDateFormat formatter = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
        String dateString = formatter.format(currentTime);
        date.setText(dateString);

        title = (EditText) view.findViewById(R.id.title);
        textContent = (EditText) view.findViewById(R.id.text_content);
        image1 = (ImageView) view.findViewById(R.id.image1);
        image2 = (ImageView) view.findViewById(R.id.image2);
        image3 = (ImageView) view.findViewById(R.id.image3);
        image4 = (ImageView) view.findViewById(R.id.image4);

        addContent = (ImageView) view.findViewById(R.id.add_content);
        addContent.setTag(201);
        addContent.setOnClickListener(this);
    }
```

```
@Override
public void onClick(View v) {
    Util util = new Util();
    switch ((Integer) v.getTag()) {
        case 201:
            addContentDialog = util.getDialog(activity, selectTypeView());
            addContentDialog.show();
            break;
        case 202:
            addContentDialog.dismiss();
            addImageDialog = util.getDialog(activity, selectWhichImageView());
            addImageDialog.show();
            break;
        case 203:
            addImageDialog.dismiss();
            String[] cameraPermissions = new String[]{
                    Manifest.permission.CAMERA};
            boolean cameraPermissionFlag = util.checkPermission(cameraPermissions,
                    activity);
            if (cameraPermissionFlag) {
                WriteDiaryFragment.this.requestPermissions(cameraPermissions,
                        Configure.CAMERA_PERMISSION_CODE);
            } else {
                openCamera();
            }
            break;
        case 204:
            addImageDialog.dismiss();
            String[] imagePermissions = new String[]{
                    Manifest.permission.READ_EXTERNAL_STORAGE,
                    Manifest.permission.WRITE_EXTERNAL_STORAGE};
            boolean imagePermissionFlag = util.checkPermission(imagePermissions,
                    activity);
            if (imagePermissionFlag) {
                WriteDiaryFragment.this.requestPermissions(imagePermissions,
                        Configure.IMAGE_PERMISSION_CODE);
            } else {
                openImageFile();
            }
            break;
        case 205:
            addContentDialog.dismiss();
```

```
                if (saveToSQLite()) {
                    Intent intent = new Intent(activity, ShowDiaryActivity.class);
                    Bundle bundle = new Bundle();
                    bundle.putSerializable("diary", diary);
                    intent.putExtras(bundle);
                    startActivity(intent);
                    activity.finish();
                }
                break;
        }
    }

    private boolean saveToSQLite() {
        diary = new Diary();
        diary.setContent(textContent.getText().toString());
        if (title.getText().toString() == null ||
                "".equals(title.getText().toString())) {
            diary.setTitle("无题");
        } else {
            diary.setTitle(title.getText().toString());
        }
        diary.setDate(date.getText().toString());
        diary.setAddress(address.getText().toString());
        diary.setAuthor(activity.getSharedPreferences("user",
                Context.MODE_PRIVATE).getString("name", ""));
        diary.setUri(uriList);
        OpenHelper openHelper = new OpenHelper(activity);
        SQLiteDatabase sqLiteDatabase = openHelper.getReadableDatabase();
        DiaryDao diaryDao = new DiaryDao(sqLiteDatabase);
        boolean flag = diaryDao.insert(diary);
        sqLiteDatabase.close();
        return flag;
    }

    private void addImageToLinearLayout(Bitmap bitmap) {
        switch (imageCount) {
            case 0:
                image1.setVisibility(View.VISIBLE);
                image1.setImageBitmap(bitmap);
                imageCount++;
                break;
            case 1:
                image2.setVisibility(View.VISIBLE);
```

```
                image2.setImageBitmap(bitmap);
                imageCount++;
                break;
            case 2:
                image3.setVisibility(View.VISIBLE);
                image3.setImageBitmap(bitmap);
                imageCount++;
                break;
            case 3:
                image4.setVisibility(View.VISIBLE);
                image4.setImageBitmap(bitmap);
                imageCount++;
                break;
        }
    }

    @Override
    public void onRequestPermissionsResult(int requestCode, String[] permissions, int[] grantResults) {
        super.onRequestPermissionsResult(requestCode, permissions, grantResults);
        if (grantResults[0] == PackageManager.PERMISSION_GRANTED) {
            Toast.makeText(activity, "获取权限成功",
                    Toast.LENGTH_SHORT).show();
            switch (requestCode) {
                case Configure.LOCATION_PERMISSION_CODE:
                    locationUtil.getLocation(this);
                    break;
                case Configure.CAMERA_PERMISSION_CODE:
                    openCamera();
                    break;
                case Configure.IMAGE_PERMISSION_CODE:
                    openImageFile();
                    break;
            }
        } else {
            Toast.makeText(activity, "获取权限失败",
                    Toast.LENGTH_SHORT).show();
        }
    }

    private void openImageFile() {
        Intent intent = new Intent();
        intent.addCategory(Intent.CATEGORY_OPENABLE);
        intent.setType("image/*");
```

```
        if (Build.VERSION.SDK_INT < 19) {
            intent.setAction(Intent.ACTION_GET_CONTENT);
        } else {
            intent.setAction(Intent.ACTION_OPEN_DOCUMENT);
        }
        startActivityForResult(intent, Configure.IMAGE_ALBUM);
    }

    private void openCamera() {
        Intent intent = new Intent("android.media.action.IMAGE_CAPTURE");
        startActivityForResult(intent, Configure.IMAGE_CAMERA);
    }

    @Override
    public void onActivityResult(int requestCode, int resultCode, Intent data) {
        super.onActivityResult(requestCode, resultCode, data);
        if (resultCode == activity.RESULT_OK) {
            switch (requestCode) {
                case Configure.IMAGE_CAMERA:
                    setBitmap(data, Configure.IMAGE_CAMERA);
                    break;
                case Configure.IMAGE_ALBUM:
                    setBitmap(data, Configure.IMAGE_ALBUM);
                    break;
            }
        }
    }

    private void setBitmap(Intent data, int from) {
        Uri uri = data.getData();
        Bitmap bitmap = null;
        if (uri == null) {
            Bundle bundle = data.getExtras();
            if (bundle != null) {
                bitmap = (Bitmap) bundle.get("data");
            }
        } else {
            try {
                bitmap = MediaStore.Images.Media.
                                getBitmap(activity.getContentResolver(), uri);
            } catch (IOException e) {
                e.printStackTrace();
            }
```

```
            }
            try {
                bitmap = MediaStore.Images.Media.getBitmap(activity.getContentResolver(),
saveBitmap(bitmap));
            } catch (IOException e) {
                e.printStackTrace();
            }
            addImageToLinearLayout(bitmap);
        }

        private Uri saveBitmap(Bitmap bitmap) {
            File f = new File(getContext().getFilesDir(), System.currentTimeMillis() + ".png");
            if (f.exists()) {
                f.delete();
            }
            try {
                FileOutputStream out = new FileOutputStream(f);
                bitmap.compress(Bitmap.CompressFormat.PNG, 90, out);
                out.flush();
                out.close();
            } catch (FileNotFoundException e) {
                e.printStackTrace();
            } catch (IOException e) {
                e.printStackTrace();
            }
            uriList += f.getPath() + ";";
            return Uri.fromFile(f);
        }

        private View selectTypeView() {
            LinearLayout typeLayout = new LinearLayout(activity);
            typeLayout.setOrientation(LinearLayout.HORIZONTAL);
            typeLayout.setGravity(Gravity.CENTER);

            if (imageCount < 4) {
                Button iamgeButton = new Button(activity);
                iamgeButton.setText("插入图片");
                iamgeButton.setTag(202);
                iamgeButton.setOnClickListener(this);
                typeLayout.addView(iamgeButton);
            }

            Button complete = new Button(activity);
```

```
            complete.setText("完成日记");
            complete.setTag(205);
            complete.setOnClickListener(this);
            typeLayout.addView(complete);
            return typeLayout;
        }


        private View selectWhichImageView() {
            LinearLayout typeLayout = new LinearLayout(activity);
            typeLayout.setOrientation(LinearLayout.HORIZONTAL);
            typeLayout.setGravity(Gravity.CENTER);

            Button textButton = new Button(activity);
            textButton.setText("从相机获取");
            textButton.setTag(203);
            textButton.setOnClickListener(this);
            Button iamgeButton = new Button(activity);
            iamgeButton.setText("从相册获取");
            iamgeButton.setTag(204);
            iamgeButton.setOnClickListener(this);

            typeLayout.addView(textButton);
            typeLayout.addView(iamgeButton);
            return typeLayout;
        }
    }
```

在 Fragment 中，先在 onCreateView()方法中调用 initView()方法，获取到相关的控件并初始化数据，同时添加点击事件。然后在 onResume()方法中调用 initLocation()方，获取定位信息，同时通过 Handler 机制将位置显示到 UI 上。在点击事件中，addContent 这个 ImageView 用来触发完成日记或者添加图片的事件，并通过 Dialog 来显示。当选择添加图片时，使用 Dialog 询问需要选择何种方式获取图片，这里的 View 使用 selectWhichImageView()方法实现。这里提供从相机获取和从图库获取两种方式，如果读者对此不熟悉，可以重新学习本书中多媒体一章。添加照片后，将图片的 Uri 保存到 Diary 中。当选择完成日记时，会调用 saveToSQLite()方法将数据保存到数据库，并跳转到 ShowDiaryActivity 中，同时将 Diary 对象传递过去。

15.3.2 日记查询

日记查询的功能分为两部分，一部分是使用 ListDiaryFragment 以列表的形式来展示日记；另一部分是当点击 ListDiaryFragment 中的某条日记时跳转到 ShowDiaryActivity 类中，显示该条日记的具体内容。这里的 ListDiaryFragment 在 fragment 包下，布局文件 list_diary.xml 如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <ListView
        android:id="@+id/diary_item"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"></ListView>
</LinearLayout>
```

这里很简单，只是使用了一个 ListView。ListDiaryFragment 的代码如下：

```
package com.buaa.diary.fragment;

import android.content.DialogInterface;
import android.content.Intent;
import android.database.sqlite.SQLiteDatabase;
import android.os.Bundle;
import android.support.v4.app.Fragment;
import android.support.v7.app.AlertDialog;
import android.support.v7.app.AppCompatActivity;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.AdapterView;
import android.widget.ListView;
import android.widget.SimpleAdapter;

import com.buaa.diary.R;
import com.buaa.diary.activity.MainActivity;
import com.buaa.diary.activity.ShowDiaryActivity;
import com.buaa.diary.database.DiaryDao;
import com.buaa.diary.database.OpenHelper;
import com.buaa.diary.entity.Diary;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

public class ListDiaryFragment extends Fragment implements AdapterView.OnItemLongClickListener {
    private static AppCompatActivity activity;
    private List<Map<String, String>> dataList;
    private SimpleAdapter simpleAdapter;
```

```
    private OpenHelper openHelper;
    private List<Diary> diaryList;

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        initData();
        View view = inflater.inflate(R.layout.list_diary, container, false);
        initView(view);
        return view;
    }

    private void initView(View view) {
        ListView listView = (ListView) view.findViewById(R.id.diary_item);
        simpleAdapter = new SimpleAdapter(
                activity,
                dataList,
                R.layout.diary_item,
                new String[]{"title", "date"},
                new int[]{R.id.item_title, R.id.item_date});
        listView.setAdapter(simpleAdapter);

        listView.setOnItemClickListener(new AdapterView.OnItemClickListener() {
            @Override
            public void onItemClick(AdapterView<?> parent, View view,
                    int position, long id) {
                Intent intent = new Intent(activity, ShowDiaryActivity.class);
                Bundle bundle = new Bundle();
                bundle.putSerializable("diary", diaryList.get(position));
                intent.putExtras(bundle);
                activity.startActivity(intent);
                activity.finish();
            }
        });

        listView.setOnItemLongClickListener(this);
    }

    private void initData() {
        openHelper = new OpenHelper(activity);
        SQLiteDatabase sqLiteDatabase = openHelper.getReadableDatabase();
        DiaryDao diaryDao = new DiaryDao(sqLiteDatabase);
        diaryList = diaryDao.queryAll();
        sqLiteDatabase.close();

        dataList = new ArrayList<>();
```

```
            for (Diary diary : diaryList) {
                Map<String, String> map = new HashMap<>();
                map.put("title", diary.getTitle());
                map.put("date", diary.getDate());
                dataList.add(map);
            }
        }

        public static ListDiaryFragment newInstance(MainActivity activity) {
            ListDiaryFragment.activity = activity;

            ListDiaryFragment fragment = new ListDiaryFragment();
            return fragment;
        }

        @Override
        public boolean onItemLongClick(AdapterView<?> parent,
                             View view, int position, long id) {
            final int location = position;
            new AlertDialog.Builder(activity)
                    .setTitle("警告")
                    .setMessage("确定删除此条数据吗？")
                    .setPositiveButton("是", new DialogInterface.OnClickListener() {
                        @Override
                        public void onClick(DialogInterface dialog, int which) {
                            SQLiteDatabase sqLiteDatabase = openHelper.getReadableDatabase();
                            DiaryDao diaryDao = new DiaryDao(sqLiteDatabase);
                            diaryDao.delete(diaryList.get(location));
                            diaryList.remove(location);
                            sqLiteDatabase.close();
                            dataList.remove(location);
                            simpleAdapter.notifyDataSetChanged();
                        }
                    })
                    .setNegativeButton("否", null)
                    .show();
            return true;
        }
    }
```

其实这里的代码和逻辑在之前都已经讲过，通过 intiData()方法调用查询数据库的方法，然后通过 ListView 展示出来，并设置 ListView 的点击事件和长按事件：当点击时跳转到 ShowDiaryActivity 中，当长按时提示是否删除该条记录。另外，为了更好地使用 ListView 展示数据，还应新建一个 item 的布局文件 diary_item.xml，代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal">

    <TextView
        android:id="@+id/item_title"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:gravity="center"
        android:paddingBottom="10dp"
        android:paddingTop="10dp"
        android:textSize="20dp" />

    <TextView
        android:id="@+id/item_date"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_weight="3"
        android:gravity="center"
        android:paddingBottom="10dp"
        android:paddingTop="10dp"
        android:textSize="20sp" />
</LinearLayout>
```

布局中使用两个 TextView 来展示日记的标题和写日记的时间。完成这些之后，在 activity 包下新建一个 ShowDiaryActivity 类，用于展示具体日记，布局文件 activity_show_diary.xml 如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <TextView
        android:id="@+id/show_title"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:layout_marginTop="30dp"
        android:textSize="18sp" />
```

```
<TextView
    android:id="@+id/show_date"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center"
    android:textSize="16sp" />

<TextView
    android:id="@+id/show_address"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center"
    android:textSize="14sp" />

<View
    android:layout_width="match_parent"
    android:layout_height="10dp" />

<ScrollView
    android:layout_width="match_parent"
    android:layout_height="0dp"
    android:layout_weight="1">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical">

        <TextView
            android:id="@+id/show_text_content"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:textSize="16sp" />

        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:orientation="horizontal">

            <ImageView
                android:id="@+id/show_image1"
                android:layout_width="0dp"
```

```
                android:layout_height="200dp"
                android:layout_weight="1"
                android:visibility="gone" />

            <ImageView
                android:id="@+id/show_image2"
                android:layout_width="0dp"
                android:layout_height="200dp"
                android:layout_weight="1"
                android:visibility="gone" />
        </LinearLayout>

        <View
            android:layout_width="match_parent"
            android:layout_height="10dp" />

        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:orientation="horizontal">

            <ImageView
                android:id="@+id/show_image3"
                android:layout_width="0dp"
                android:layout_height="200dp"
                android:layout_weight="1"
                android:visibility="gone" />

            <ImageView
                android:id="@+id/show_image4"
                android:layout_width="0dp"
                android:layout_height="200dp"
                android:layout_weight="1"
                android:visibility="gone" />

        </LinearLayout>
    </LinearLayout>
</ScrollView>

<Button
    android:id="@+id/go_to_list"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
```

```
        android:text="返回日记列表" />
</LinearLayout>
```

如果读者仔细研究，就会发现这个布局和记录日记的 Fragment 的布局文件 write_diary.xml 比较相似，只是将 write_diary.xml 中的 EditText 部分修改为 TextView 部分了，同时将最底部的 ImageView 修改为 Button 按钮。完成布局的修改之后，修改 ShowDiaryActivity 类来将数据展示到 UI 上，代码如下：

```
package com.buaa.diary.activity;

import android.content.Intent;
import android.graphics.Bitmap;
import android.net.Uri;
import android.provider.MediaStore;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.KeyEvent;
import android.view.View;
import android.widget.Button;
import android.widget.ImageView;
import android.widget.TextView;

import com.buaa.diary.R;
import com.buaa.diary.entity.Diary;
import com.buaa.diary.util.Configure;

import java.io.File;
import java.io.IOException;

public class ShowDiaryActivity extends AppCompatActivity {

    private Diary diary;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_show_diary);
        initData();
        initView();
    }

    private void initData() {
        Bundle bundle = getIntent().getExtras();
        diary = (Diary) bundle.getSerializable("diary");
```

```
    }

    private void initView() {
        if (diary == null) {
            return;
        }
        TextView showTitle = (TextView) findViewById(R.id.show_title);
        showTitle.setText(diary.getTitle());
        TextView showAddress = (TextView) findViewById(R.id.show_address);
        showAddress.setText(diary.getAddress());
        TextView showDate = (TextView) findViewById(R.id.show_date);
        showDate.setText(diary.getDate());
        TextView showTextContent = (TextView) findViewById(R.id.show_text_content);
        showTextContent.setText(diary.getContent());

        ImageView image1 = (ImageView) findViewById(R.id.show_image1);
        ImageView image2 = (ImageView) findViewById(R.id.show_image2);
        ImageView image3 = (ImageView) findViewById(R.id.show_image3);
        ImageView image4 = (ImageView) findViewById(R.id.show_image4);
        ImageView[] imageViews = new ImageView[]{image1, image2, image3, image4};

        String uris = diary.getUri();
        if (uris != null && !"".equals(uris)) {
            String[] uriArr = uris.split(";");
            for (int i = 0; i < uriArr.length; i++) {
                Bitmap bitmap = null;
                try {
                    bitmap = MediaStore.Images.Media.getBitmap(
                    getContentResolver(),
                    Uri.fromFile(new File(uriArr[i])));
                } catch (IOException e) {
                    e.printStackTrace();
                }
                imageViews[i].setImageBitmap(bitmap);
                imageViews[i].setVisibility(View.VISIBLE);
            }
        }

        Button goBackButton = (Button) findViewById(R.id.go_to_list);
        goBackButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                goMainActivity();
```

```
                }
            });
        }

        @Override
        public boolean onKeyDown(int keyCode, KeyEvent event) {
            if (keyCode == KeyEvent.KEYCODE_BACK) {
                goMainActivity();
            }
            return true;
        }

        private void goMainActivity() {
            Intent intent = new Intent(ShowDiaryActivity.this, MainActivity.class);
            intent.putExtra("from", Configure.FROM_SHOW_DIARY_ACTIVITY);
            startActivity(intent);
            finish();
        }
    }
```

不管是在 WriteDiaryFragment 中完成日记还是在 ListDiaryFragment 中通过点击 item，最终都会进入 ShowDiaryActivity 中，并将刚记录的或者选中的 Diary 对象传递过来。所以在 ShowDiaryActivity 中，首先通过 intiData()方法获取传递过来的 Diary 对象，然后通过 initView 方法将 Diary 对象中的数据设置到控件中。另外，这里也设置了手机的回退事件处理方法。

15.3.3 个人中心

在本应用中，个人中心是通过 PersonFragment 来实现的，也在 fragment 包下。这里允许读者对生日、职业、爱好、个人说明进行修改，布局文件 person.xml 如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="50dp"
        android:gravity="center"
        android:orientation="horizontal">

        <TextView
```

```
            android:id="@+id/person_name"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:textSize="22sp" />
    </LinearLayout>

    <View
        android:layout_width="match_parent"
        android:layout_height="1sp"
        android:layout_marginTop="30dp"
        android:background="@color/colorAccent" />

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:gravity="center_vertical"
        android:orientation="horizontal">

        <TextView
            android:layout_width="0dp"
            android:layout_height="60dp"
            android:layout_weight="2"
            android:gravity="center"
            android:text="出生日期"
            android:textSize="20sp" />

        <TextView
            android:id="@+id/person_date"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_weight="5"
            android:textSize="20sp" />
    </LinearLayout>

    <View
        android:layout_width="match_parent"
        android:layout_height="1sp"
        android:background="@color/colorAccent" />

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:gravity="center_vertical"
```

```
        android:orientation="horizontal">

        <TextView
            android:layout_width="0dp"
            android:layout_height="60dp"
            android:layout_weight="2"
            android:gravity="center"
            android:text="职业"
            android:textSize="20sp" />

        <TextView
            android:id="@+id/person_job"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_weight="5"
            android:textSize="20sp" />
    </LinearLayout>

    <View
        android:layout_width="match_parent"
        android:layout_height="1sp"
        android:background="@color/colorAccent" />

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:gravity="center_vertical"
        android:orientation="horizontal">

        <TextView
            android:layout_width="0dp"
            android:layout_height="60dp"
            android:layout_weight="2"
            android:gravity="center"
            android:text="兴趣爱好"
            android:textSize="20sp" />

        <TextView
            android:id="@+id/person_love"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_weight="5"
            android:textSize="20sp" />
```

```
        </LinearLayout>

        <View
            android:layout_width="match_parent"
            android:layout_height="1sp"
            android:background="@color/colorAccent" />

        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:gravity="center_vertical"
            android:orientation="horizontal">

            <TextView
                android:layout_width="0dp"
                android:layout_height="60dp"
                android:layout_weight="2"
                android:gravity="center"
                android:text="个人说明"
                android:textSize="20sp" />

            <TextView
                android:id="@+id/person_construction"
                android:layout_width="0dp"
                android:layout_height="wrap_content"
                android:layout_weight="5"
                android:textSize="20sp" />
        </LinearLayout>

        <View
            android:layout_width="match_parent"
            android:layout_height="1sp"
            android:background="@color/colorAccent" />
    </LinearLayout>
```

代码很容易理解，只是简单地罗列了几个 TextView，用于展示数据。下面在 PersonFragment 中获取控件并设置值，代码如下：

```
package com.buaa.diary.fragment;

import android.app.AlertDialog;
import android.content.Context;
import android.content.DialogInterface;
import android.content.SharedPreferences;
```

```
import android.os.Bundle;
import android.support.v4.app.Fragment;
import android.support.v7.app.AppCompatActivity;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.EditText;
import android.widget.TextView;
import android.widget.Toast;

import com.buaa.diary.R;
import com.buaa.diary.activity.MainActivity;


public class PersonFragment extends Fragment implements View.OnClickListener,
View.OnLongClickListener {

    private static AppCompatActivity activity;
    private SharedPreferences sharedPreferences;
    private TextView personDate;
    private TextView personJob;
    private TextView personLove;
    private TextView personConstruction;

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        View view = inflater.inflate(R.layout.person, container, false);
        initView(view);
        return view;
    }

    private void initView(View view) {
        sharedPreferences = activity.getSharedPreferences("user",
                Context.MODE_PRIVATE);
        TextView nameText = (TextView) view.findViewById(R.id.person_name);
        nameText.setText(sharedPreferences.getString("name", "") + "简介");

        personDate = (TextView) view.findViewById(R.id.person_date);
        personJob = (TextView) view.findViewById(R.id.person_job);
        personLove = (TextView) view.findViewById(R.id.person_love);
        personConstruction = (TextView) view.findViewById(R.id.person_construction);
```

```
        personDate.setTag("出生日期");
        personJob.setTag("职业");
        personLove.setTag("兴趣爱好");
        personConstruction.setTag("个人说明");

        personDate.setText(sharedPreferences.getString("date", ""));
        personJob.setText(sharedPreferences.getString("job", ""));
        personLove.setText(sharedPreferences.getString("love", ""));
        personConstruction.setText(sharedPreferences.getString("construction", ""));

        personDate.setOnClickListener(this);
        personJob.setOnClickListener(this);
        personLove.setOnClickListener(this);
        personConstruction.setOnClickListener(this);

        personDate.setOnLongClickListener(this);
        personJob.setOnLongClickListener(this);
        personLove.setOnLongClickListener(this);
        personConstruction.setOnLongClickListener(this);
    }

    @Override
    public void onClick(View v) {
        Toast.makeText(getContext(), "长按可修改内容", Toast.LENGTH_LONG).show();
    }

    @Override
    public boolean onLongClick(View v) {
        switch (v.getId()) {
            case R.id.person_date:
                dialogForChangeContent("date", personDate);
                break;
            case R.id.person_job:
                dialogForChangeContent("job", personJob);
                break;
            case R.id.person_love:
                dialogForChangeContent("love", personLove);
                break;
            case R.id.person_construction:
                dialogForChangeContent("construction", personConstruction);
                break;
        }
        return true;
```

```
    }

    private void dialogForChangeContent(final String key, final TextView text) {
        final EditText changeText = new EditText(activity);
        AlertDialog.Builder builder = new AlertDialog.Builder(activity);
        builder.setCancelable(false);
        builder.setTitle("修改" + text.getTag());
        builder.setView(changeText);
        builder.setPositiveButton("确定", new DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int which) {
                String value = changeText.getText().toString();
                if (value != null && !"".equals(value)) {
                    text.setText(value);
                    SharedPreferences.Editor editor = sharedPreferences.edit();
                    editor.putString(key, value);
                    editor.commit();
                }
            }
        });
        builder.setNegativeButton("取消", null);
        builder.show();
    }

    public static PersonFragment newInstance(MainActivity activity) {
        PersonFragment.activity = activity;
        PersonFragment fragment = new PersonFragment();
        return fragment;
    }
}
```

PersonFragment 主要通过 initView 来获取各控件，然后从 ShardPreferences 中获取值并赋给对应的 TextView，同时设置 TextView 的长按事件和点击事件。当长按时可以修改控件的值并保存到 ShardPreferences 中。

15.3.4 AndroidManifest.xml 及其他配置文件

完成上述开发之后，只需要修改 AndroidManifest.xml 文件就可以完成应用的开发了。AndroidManifest.xml 文件代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.buaa.diary">
```

```
        <uses-permission android:name="android.permission.INTERNET" />
        <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
        <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
        <uses-permission android:name="android.permission.CAMERA" />
        <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
        <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />

        <application
            android:allowBackup="true"
            android:icon="@mipmap/ic_launcher"
            android:label="@string/app_name"
            android:supportsRtl="true"
            android:theme="@style/Theme.AppCompat.Light.NoActionBar">
            <activity android:name=".activity.LoginActivity">
                <intent-filter>
                    <action android:name="android.intent.action.MAIN" />

                    <category android:name="android.intent.category.LAUNCHER" />
                </intent-filter>
            </activity>
            <activity android:name=".activity.MainActivity" />
            <activity android:name=".activity.ShowDiaryActivity"></activity>
        </application>
    </manifest>
```

这里添加了应用需要的权限，以及系统生成的各个 Activity 的配置，其他未做修改。在应用安装后，我们希望应用名称显示为“口袋日记”，所以修改 res 目录下 values 文件夹中的 strings.xml 文件如下：

```
<?xml version="1.0" encoding="utf-8"?>
     <resources>
         <string name="app_name">口袋日记</string>
    </resources>
```

在应用中还使用到了几种颜色，所以修改 res 目录下 values 文件夹中的 colors.xml 文件如下：

```
<?xml version="1.0" encoding="utf-8"?>
    <resources>
        <color name="colorPrimary">#3F51B5</color>
        <color name="colorPrimaryDark">#303F9F</color>
        <color name="colorAccent">#FF4081</color>
    </resources>
```

到这里，整个应用就开发完成了。运行程序，设置账号，如图 15-2 所示。设置完账号后

正确输入账号才能够进入写日记的界面，这里会默认给出一个标题，并定位出所在位置以及时间，如图 15-3 所示。

在这个界面中，我们可以通过点击“加号”按钮完成日记或者添加图片的操作。添加图片时，图片既可以来自手机相册，也可以来自相机，如图 15-4 所示。



图 15-2　设置账号



图 15-3 编写日记



图 15-4　向日记中添加图片

写完日记，点击保存日记之后会进入日记中心界面，这里展示的是日记列表，如图 15-5 所示。

如果此时我们点击某条日记，就能够看到这条日记的具体内容，如图 15-6 所示。



图 15-5 日记中心



图 15-6 日记详情

在日记详情界面中，我们可以返回日记列表，然后继续写日记。除了写日记与看日记之外，还可以进入个人中心，展示一些信息，如图 15-7 所示。

个人中心里面的信息也是可以编辑的，想要修改个人说明时，只需要长按即可。图 15-8 就是长按“兴趣爱好”条目之后的场景。



图 15-7 个人中心



图 15-8 修改个人中心信息

15.4 将应用打包并发布到小米应用商店

完成应用的开发之后，本节我们就来讲解如何将应用打包并将其发布应用。

15.4.1 应用打包

所谓应用打包，就是将开发的应用打包成 APK 文件。可能有读者会问，我们之前在开发完一个 Android 应用之后，直接点击运行，选择一个模拟器或者真机就完成了应用安装，这说明 Android Studio 已经默认给我们生成了一个 APK 文件，为什么还要自己打包 APK 文件呢？

确实如此，Android Studio 已经默认给我们生成了一个 APK 文件，在 app/build/outputs/apk 文件夹下，如图 15-9 所示。

如果读者在该文件夹下找不到，只需要在工具栏中的“Build”下点击“Build APK”即可，如图 15-10 所示。

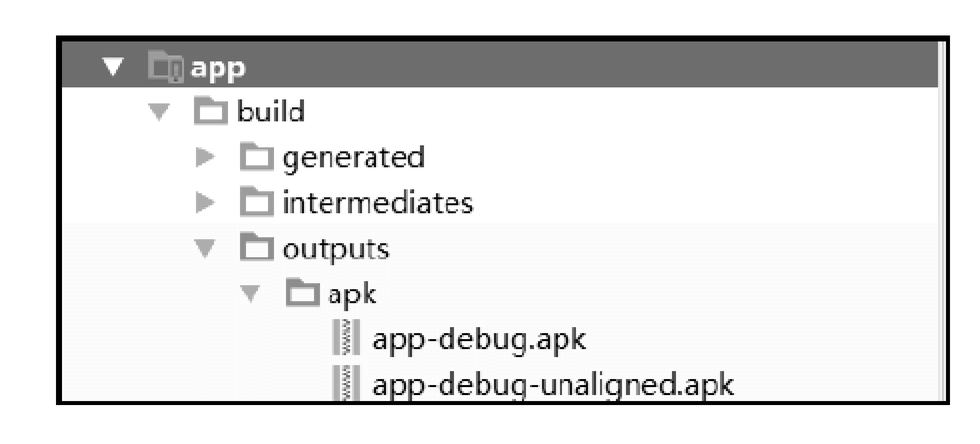


图 15-9 debug 版本的 APK 文件所在位置

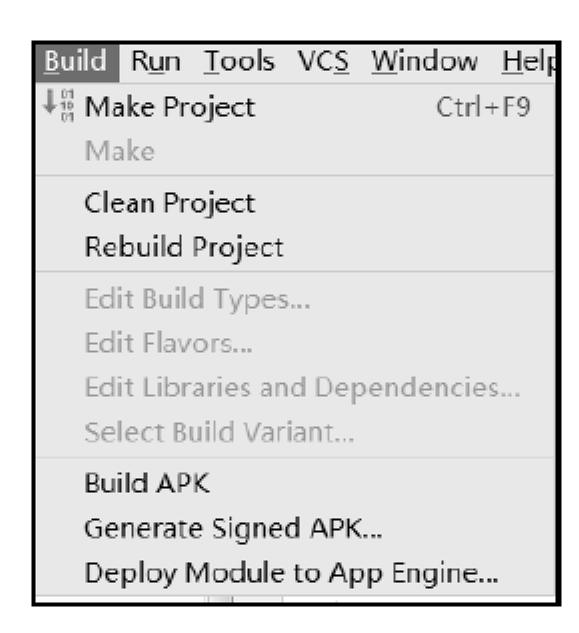


图 15-10 如何生成 debug 版本的 APK 文件

通过观察 app-debug.apk（app-debug-unaligned.apk 是打包时产生的中间文件）这个 Android Studio 默认生成的 APK，我们会发现它的名字里带有“debug”字样，意味着调试版本的 APK 文件。由此，读者可能会想到在讲解地理信息技术时我们使用的调试版的签名文件 debug.keystore。这两者直接有很深的联系，Android Studio 默认生成的正是使用签名文件 debug.keystore 来打包 APK 的。之前的章节我们说过，签名文件对于一个 Android 应用的重要性，所以要正式发布一款应用必然需要一个正式版本的签名文件。在 Android Studio 中创建一个签名文件其实很简单，只需要执行如下几步操作即可。

步骤 01 打开工具栏中的“Build”工具，点击“Generate Signed APK”，如图 15-11 所示。

步骤 02 弹出如图 15-12 所示的界面。我们需要在这里填入正式版的签名文件，也就是 Key store。

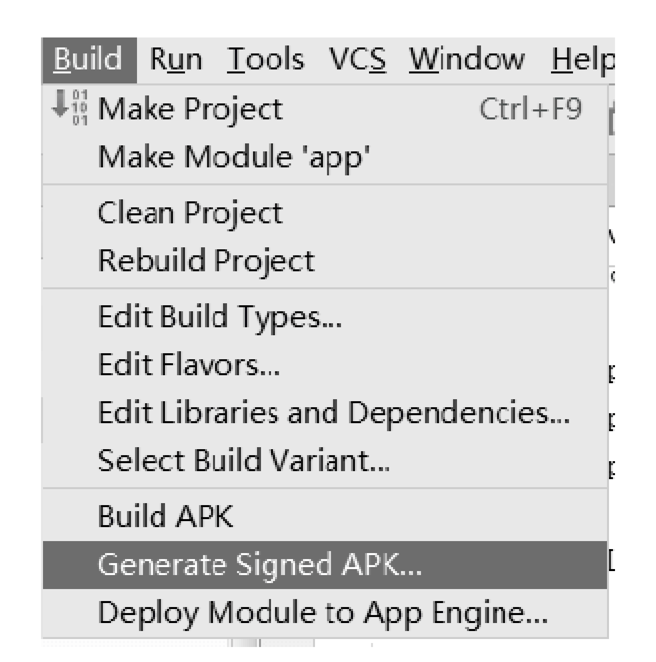


图 15-11　选择“Generate Signed APK”来打包正式版 APK

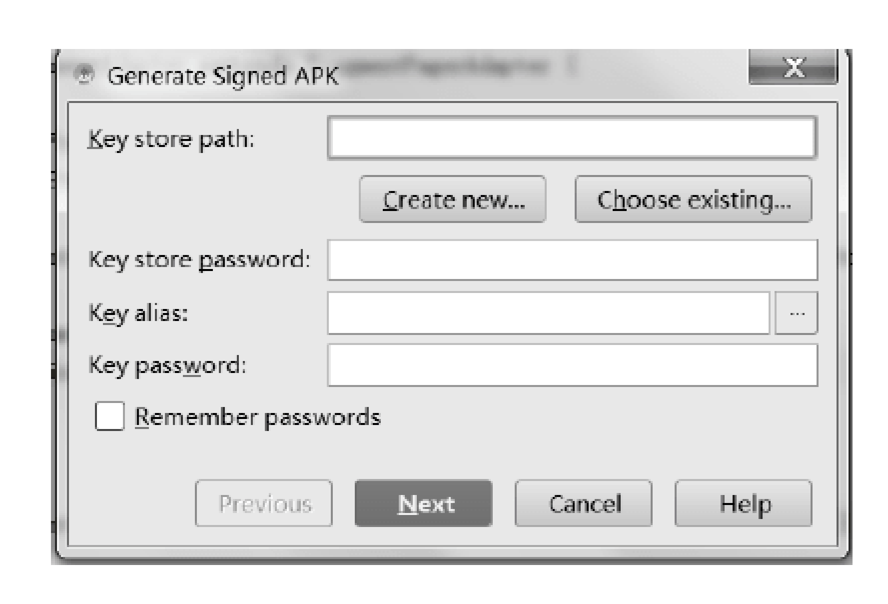


图 15-12　填入打包所需的正式签名文件

步骤 03 如果是第一次使用签名就单击“Create new”按钮创建一个新的签名文件。如果之前有过签名文件，就选择蓝框部分进行导入。这里单击“Create new”按钮，出现如图 15-13 所示的界面，然后填写相关的选项。

步骤 04 单击“OK”按钮，会回到第二步的界面，只不过此时界面中的内容自动显示出来了，效果如图 15-14 所示。

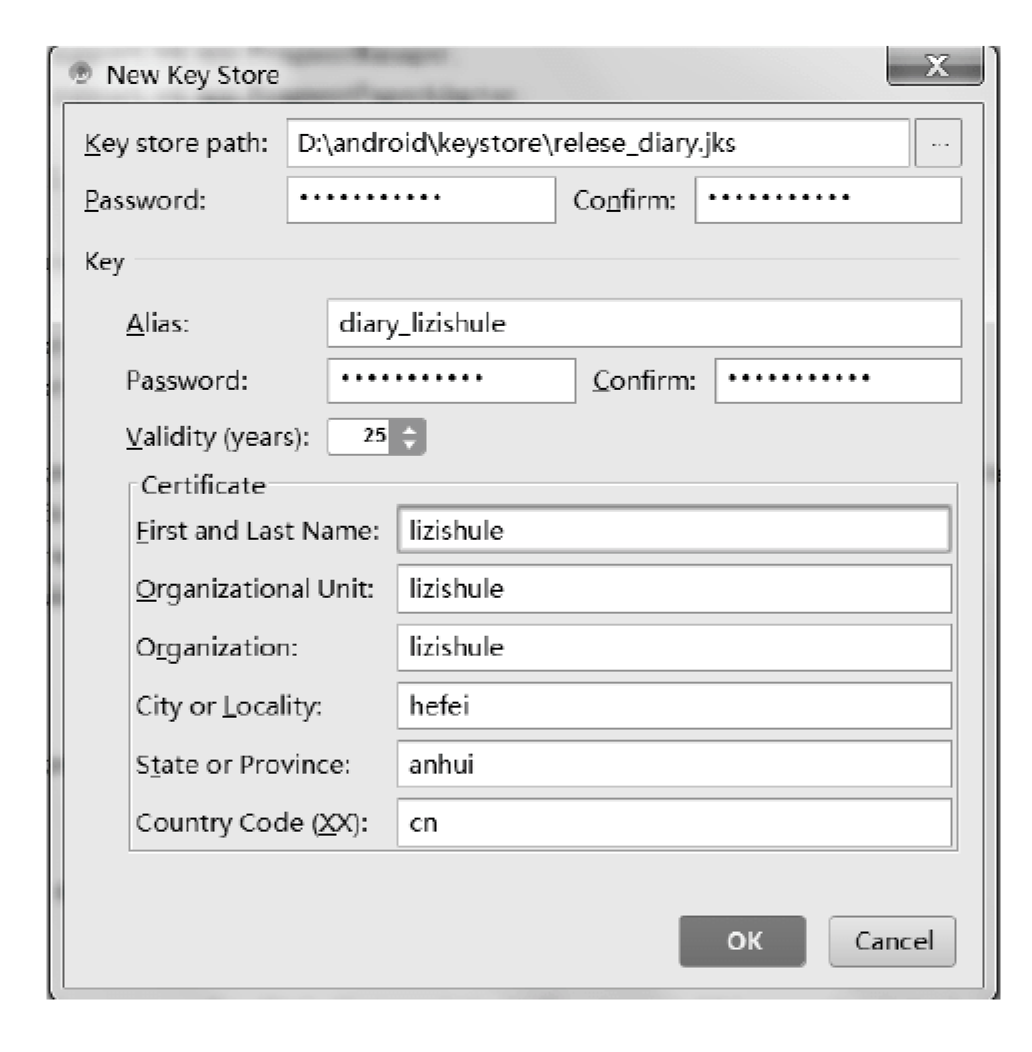


图 15-13　生成签名文件

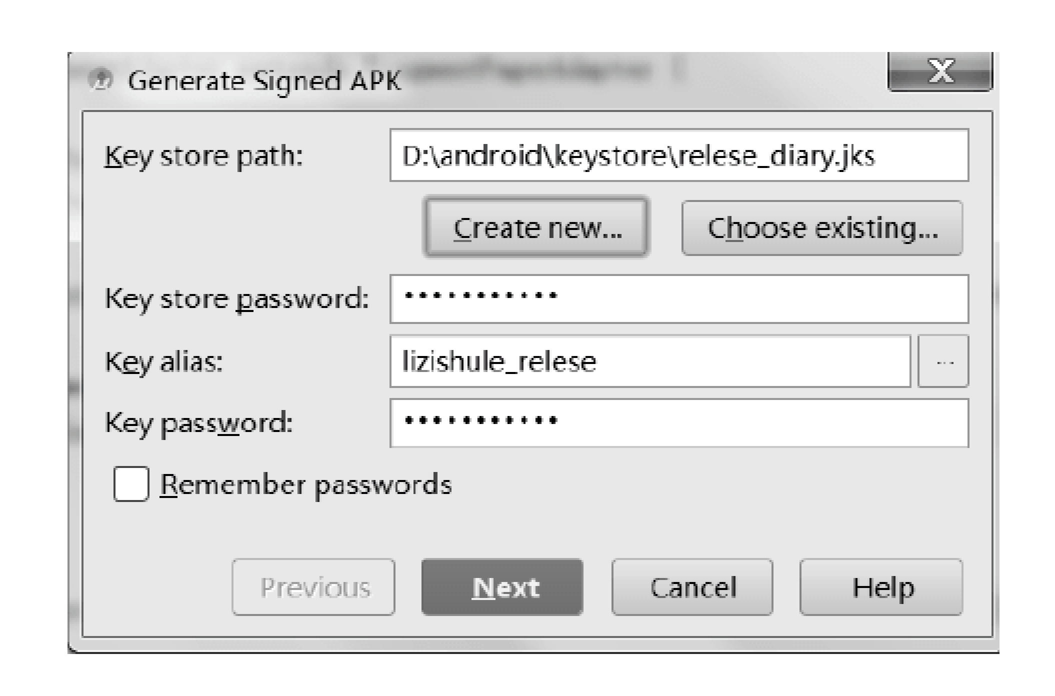


图 15-14　Studio 自动回填生成的签名文件

步骤 05 单击“Next”按钮会提示输入密码，这里的密码就是我们在创建签名文件时所输入的密

码，如图 15-15 所示。

步骤 06 单击“OK”按钮，进入如图 15-16 所示的界面，这是打包过程的最后一步操作。



图 15-15　打包之前进行校验

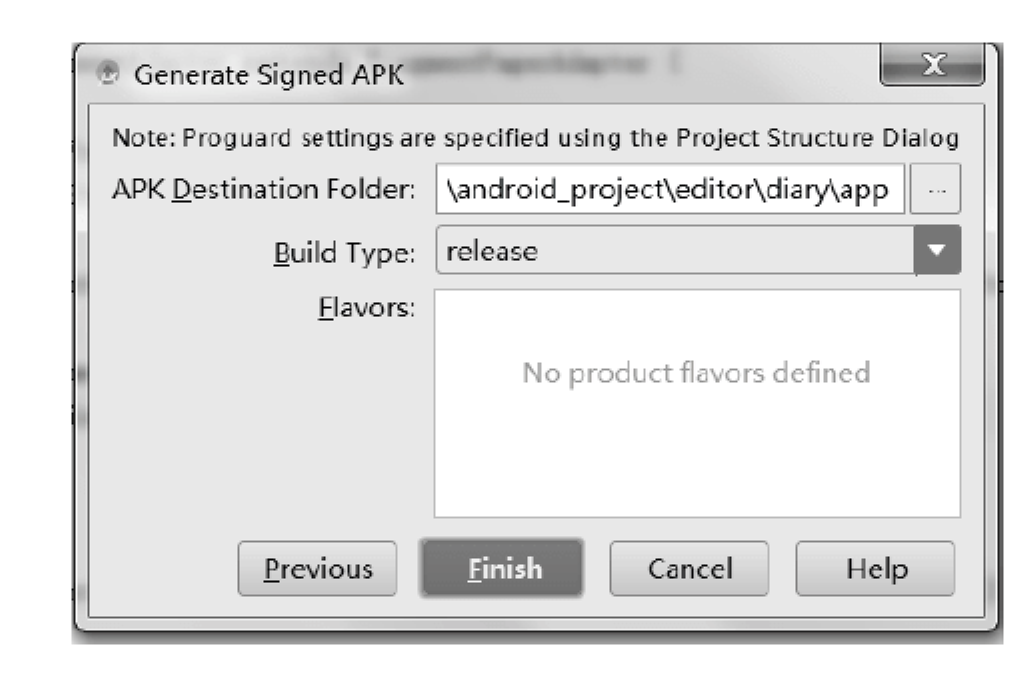


图 15-16　最后确认是否打包 APK 文件

步骤 07 单击“Finish”按钮就可以打包正式版的 APK 了，将在 Android Studio 底部出现 Gradle 正在执行 assembleRelease 任务的信息，如图 15-17 所示，这说明我们的操作是正确的，Studio 正在打包。



图 15-17　Gradle 在后台进行打包

其实，在第四步完成之后，一个签名文件就创建完成了，后面的步骤都是用来签名 APK 的。执行完这 7 步之后，app 目录下出现 app-release.apk 文件，如图 15-18 所示。

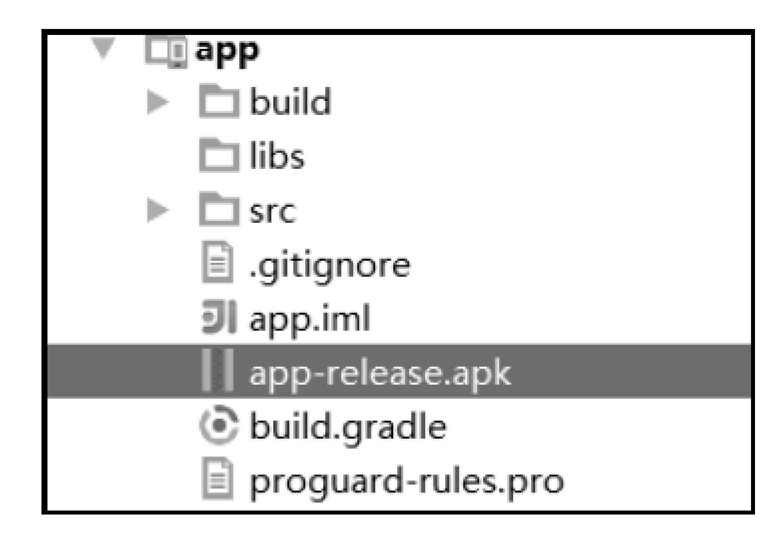


图 15-18　打包完成后 APK 文件所在的位置

如果此时安装应用，就会发现图片依旧是 Android 默认的机器人图标。应用的图片是由 AndroidManifest.xml 文件中 Application 的 android:icon="@mipmap/ic_launcher"决定的，默认的是 mipmap 下的 ic_launcher.png 图片。这里可以通过修改 mipmap 下的 ic_launcher.png 图片来改变应用显示的图标。

15.4.2　发布应用到小米应用商店

一个正式版的应用已经打包完成，接下来就要发布到应用市场了。在国内，应用市场很多，呈现群雄逐鹿之势，知名的有应用宝、360 手机助手、小米应用商店、华为应用商店、百度手机助手、91 手机助手、豌豆荚（这里是按照 2016 年第一季度各大应用市场日活数排名的）。这里以小米应用商店为例来讲解如何发布一个应用到应用商店，而且其他几种应用市场的发布方式与小米应用商店大同小异，读者可自行尝试。

要在小米应用商店发布一个应用，需要如下几个步骤。

步骤 01 进入“http://app.mi.com/”网站，单击导航栏中的“开发者”选项，进入选择开发者类型的界面，如图 15-19 所示，这里可根据具体需求来选择。另外，这一步需要注册小米账号。



图 15-19　选择开发者类型

步骤 02 选择开发者账号类型之后就可以开通开发者账号了，这里需要填写基本信息和详细资料，如图 15-20 所示。完成之后，需要等待资料审核，这个过程不会很长，审核结果会通过邮箱通知你。

开通个人开发者帐号
基本信息
*邮箱：[redacted]
您可以使用邮箱登录或 重置密码
*开发者名称(昵称)：
开发者名称会显示在应用中

详细资料
*真实姓名：
*联系电话：
建议填写手机，如果是固话请输入区号
*身份证号码
*身份证扫描件 上传 本人持身份证正面照，且照片小于4MB
示例：
手臂完整露出
身份证正面信息清晰可见
QQ：
MSN：
注册

图 15-20　填写开通开发者账号所需的信息

步骤 03 当收到资格审核通过的邮件后再次进入“http://app.mi.com/”网站，单击导航栏中的“开发者”选项会进入如图 15-21 所示的界面，选择“手机及平板应用”即可。



图 15-21 选择应用类型

步骤 04 进入创建新应用的界面，如图 15-22 所示。

【最新通知】为响应有关政府部门对金融投资理财类、彩票类和医疗健康分类产品的严格管理要求，以上类别应用需下载并提交【免责承诺函】

移动应用 - 应用管理

应用状态： 全部(0) | 已上架(0) | 定时上线(0) | 审核中(0) | 审核未通过(0) | 被下架(0) | 未发布(0) |

帮助文档： 开发者文档站 开发者常见问题 首发文档 删除/下架流程 在线客服

+ 创建新应用

图 15-22 创建新应用

步骤 05 单击“创建新应用”按钮，进入新的界面，填写相关信息，如图 15-23 所示。

创建新应用

默认语言： 简体中文

操作系统： Android iOS（仅能使用推送服务和统计服务）

应用名称： 口袋日记

请输入您的应用名称，不超过8个汉字或16个字符，以后可以修改

应用包名： com.buaa.diary

请注意包名一旦填写不能修改，请填写一个足够复杂、不易重复的包名，以免影响发布

什么是应用包名？

创建

图 15-23 填写新应用信息

这里的包名就是 AndroidManifest.xml 文件中 package 属性的值，即“com.buaa.diary”。单击“创建”按钮之后，进入如图 15-24 所示的确认界面。



图 15-24 确认创建的新应用信息

步骤 06 单击“发布应用”按钮，进入上传应用界面，上传应用即可，效果如图 15-25 所示。



图 15-25 上传应用

步骤 07 完成上传之后，接下来只需要完善资料即可，这里不再介绍。最后提交完成后会提示等待审核，一般几天即可，如图 15-26 所示。



图 15-26 完成上传等待审核

15.5 小　结

本章通过一个完整的应用讲述了在开发实践中如何将一个产品从需求变为实际可用的应用，其并将其发布到应用市场。读者可能会发现本章中的代码较多，且没有注释，讲解也不是特别多，这是因为本章程序中所涉及的技术都在之前讲解过，如果读者系统地学习了之前的内容，阅读本章的代码应该非常容易。另外，本章中的完整应用是笔者自己开发的教学案例，并没有产品经理和 UI 设计师的配合，所以在布局以及功能上有所欠缺，读者应将重点放在开发流程以及如何打包应用和发布应用上。

至此，本书就结束了。希望读者可以通过阅读本书掌握 Android 开发中的各项技术，并且能够独立开发出一款产品。